Multigateway

USER'S MANUAL

VERSION 3.1

EXTERNAL

BM&FBOVESPA
The New Exchange

Introduction

This document introduces MEGADirect, BM&FBOVESPA's new interface for entering orders into the exchange's MEGABOLSA trading platform.

This manual should be studied by managers and employees of all participating institutions currently using the service.

Whenever necessary, BOVESPA will inform all users who have received this manual of any revisions and alterations made to the chapters.

Prior to any Multigateway access requests, please refer to the guidelines available at www.bovespanet.com.br. Regulations / Order Routing / Automated Connections.

Technical support for participating institutions will be provided by the Call Center on the São Paulo telephone number (5511) 3233-2333 or by e-mail: cab@bovespa.com.br.

Change Log

Version	Description	Author
3.1	- Added support to market to limit orders - Deprecated use of the QueryLimits function	BS
1.0	First version	JML

Contents

Chapters	Page
Introduction	2
Modifications	2
Content	4
I - GOALS, APPLICATION AND DEFINITIONS	
1 – GOALS	8
2 – APPLICATION	8
3 – DEFINITIONS	8
3.1 Terminology and Designations	8
3.2 System Rules	8
3.2.1 Order Modification	8
3.2.1.1 API	8
3.2.1.2 FIX	9
3.2.2 Cancellation or Modification of Previously Modified Order - API and FIX	9
3.3 Restrictions	9
3.4 References	9
II - PROCEDURES	
1 -API PROTOCOL	10
1.1 Connection	10
1.2 Session Layer	10
1.2.1 Message Sequencing	11
1.2.2 Resend Message Buffer	11
1.2.3 Resend Mechanism	11
1.2.4 Active Connection Check	11
1.2.5 Session Layer Messages	12
1.2.5.1 Authentication Request Message	12
1.2.5.2 Authentication Success Message	13
1.2.5.3 Authentication Error Message	13
1.2.5.4 Message Resend Request Message	14
1.2.5.5 New Message Sequence Notification Message	14

1.2.5.6 Heartbeat Message 14
1.2.5.7 Heartbeat Response Message 15

1.3 Application Layer 15
1.3.1 Index 15
1.3.2 Control Number 15
1.3.3 Resend Sequence Control 15
1.3.4 Application Layer Messages 16
1.3.4.1 New Order Registration Request Message 16
1.3.4.2 New Order Cancellation Request Message 16
1.3.4.3 Order Rerequest Message 17
1.3.4.4 Limit Query Message 17
1.3.4.5 Last Index Query Message 17
1.3.4.6 Order Book Query Message 17
1.3.4.7 Ack Message 18
1.3.4.8 Order Registration Confirmation Response Message 18
1.3.4.9 Order Cancellation Confirmation Response Message 19
1.3.4.10 Error Event Response Message 19
1.3.4.11 Order Execution Response Message 20
1.3.4.12 Index Query Response Message 20
1.3.4.13 Limit Query Response Message 21
1.3.4.14 Order Book Query Response Message 21
1.3.4.15 Order Modification Request Message 22
1.3.4.16 Order Modification Confirmation Response Message 22

1.4 DLL 23
1.4.1 GWHBClientCOM.dll Component Installation 23
1.4.2 Basic Use 24
1.4.3 Component Classes 24
1.4.3.1 HBConnection 24
1.4.3.2 HBConnection2 27
1.4.3.3 HBOrderInfo 28
1.4.3.4 HBCancelOrder 29
1.4.3.5 HBQueryLimits 29
1.4.3.6 HBRecalMsgs 30
1.4.3.7 HBOrderResponse 30
1.4.3.8 HBCancelResponse 30
1.4.3.9 HBErrorResponse 31
1.4.3.10 HBIndexResponse 32
1.4.3.11 HBLimitResponse 32
1.4.3.12 HBTradeResponse 33
1.4.3.13 HBModifyResponse 34
1.4.3.14 HBNewOrder 35
1.4.3.15 HBModifyOrder 35
1.4.3.16 HBOrderBookResponse 35
1.4.3.17 HBOrderBookResponse2 36

1.5 LIB 37
1.5.1 LibGnr.a Component Installation 37

1.5.2 Basic Use 37
1.5.3 Functions 38
1.5.3.1 GetStatus 38
1.5.3.2 GetLastCtrlNumber 38
1.5.3.3 Start 38
1.5.3.4 Stop 39
1.5.3.5 SendOrder 39
1.5.3.6 CancelOrder 41
1.5.3.7 QueryLimits 41
1.5.3.8 QueryIndex 41
1.5.3.9 RecallMessages 41
1.5.3.10 QueryOrderBook 41
1.5.3.11 ModifyOrder 42
1.5.4 Events 42
1.5.4.1 OrderResponse 43
2 1.5.4.2TradeResponse 43
1.5.4.3 CancelResponse 44
1.5.4.4 LimitResponse 44
1.5.4.5 ErrorResponse 45
1.5.4.6 TradeCancelResponse 45
1.5.4.7 IndexResponse 45
1.5.4.8 OrderBookResponse 46
1.5.4.9 AuthenticationOk 47
1.5.4.10 AuthenticationError 47
1.5.4.11 ConnectionAccepted 47
1.5.4.12 ConnectionError 47
1.5.4.13 ModifyResponse 48

2 -FIX PROTOCOL 49

2.1 FIX 49

2.2 FIX Messages 49
2.2.1 Header 49
2.2.2 Trailer 50
2.2.3 Administrator's Messages 51
2.2.3.1 Logon 51
2.2.3.2 Logout 51
2.2.3.3 HeartBeat 52
2.2.3.4 TestRequest 52
2.2.3.5 ResendRequest 52
2.2.3.6 Reject 53
2.2.3.7 SequenceReset 53
2.2.4 Application Messages 53
2.2.4.1 NewOrderSingle 54
2.2.4.2 OrderCancelRequest 55
2.2.4.3 OrderCancelReject 55
2.2.4.4 ExecutionReport 56

2.2.4.5 OrderCancelReplace 58

3 -FIX MESSAGES 59

3.1 NewOrderSingle 59
3.1.1 Order Registration 59
3.1.2 Order Registration Request Followed by Rejection 59
3.1.3 Freezing of Order 60
3.1.4 Freezing of Order Followed by Acceptance upon Defreezing 61
3.1.5 Freezing of Order Followed by Rejection upon Defreezing 62
3.1.6 Order Send Followed by Partial Trade, Freezing and Subsequent Defreezing of Order upon Acceptance 63
3.1.7 Freezing of Order Followed by Auction 64
3.1.8 New Order Send during Auction 64

3.2 OrderCancelRequest 65
3.2.1 Cancellation of a Registered Order 65
3.2.2 Cancellation of a Non-Existent Order 65
3.2.3 Cancellation of a Previously Cancelled Order 66
3.2.4 Cancellation of an Order at Auction 67

3.3 OrderCancelReplace 68
3.3.1 Modifying a Registered Order 68
3.3.2 Modifying a Non-Existent Order 68

3.4 Trade 69
3.4.1 Trade – Filled 69
3.4.2 Trade – Partially Filled 70
3.4.3 Cancel Trade 71

Goals, Application and Definitions

1 – Goals

This document is designed to provide a detailed description of the functioning and framework of *MultiGateway.* The aim is to impart a broader knowledge of the system as a whole, as well as to facilitate a greater understanding of the particularities of each constituent module. Its functionalities, features of its connections with customers and the trading system, data structures and other particulars are described in detail.

2 – Application

This manual is aimed at Broker Societies using online tools such as Home Brokers to connect online and carry out online trades.

3 – Definitions

3.1 Terminology and Designations

- **DLL**: Dynamic Link Library
- **MEGA BOLSA**: Is the trading system of the São Paulo Stock Exchange, which manages and controls information and is responsible for registering all trades executed at BOVESPA.
- **MULTIGATEWAY:** Is the Home Broker and automated connection server.
- **Pipe**: A mechanism to allow a process to transfer information to another process.
- **Linkedition**: A method to add a library to an application.
- **Callback**: A pointer to a function.
- **LIB**: A Unix library added to the client application to access functions.

3.2 System Rules

3.2.1 Order Modification

3.2.1.1 API

To modify an order, it is necessary to inform all fields from the original order even if such modification is made only to a single field.

In the "Order Quantity" field the following detail should be observed: The modified quantity refers to the remaining quantity. Thus, in the case of a partial execution, the quantity modification informed will be added to the executed quantity. Therefore, whenever a user sends a modification it will be necessary to calculate the quantity, whether or not they wish to modify it.

Example:

Original Qty: 500
Executed Qty: 100 (remaining qty: 400)

In the foregoing example, if a ModifyOrder with Qty 500 is sent, it is a clear indication that the remaining quantity – i.e. 400 – should be increased to 500. As a result, the total quantity will be 600 (600 – 100 executed = 500).

In this case, to prevent the remaining quantity from being modified, a Qty of 400 should be relaid so as to ensure that the total quantity returned is 500 (500 - 100 = 400).

For all other fields the original values can remain unchanged, unless one wishes to change them.

3.2.1.2 FIX

To modify an order, it is necessary to inform all fields from the original order even if such modification is made only to a single field.

To change an order quantity, the value sent in the quantity field should always be the total order quantity and not just the remaining quantity, as in the API case.

Ex.: Original order:

Quantity	= 500
Executed	= 300
Remaining	= 200

Modification 1

Quantity	= 400

In the foregoing modification, the remaining quantity would be 100, since 400 – 300 (which is the executed quantity) = 100.

Modification 2

Quantity	= 600

In the foregoing modification, the remaining quantity would be 300, since 600 - 300 (which is the executed quantity) = 300.

Modification 3

Quantity	= 200

In this case, an error would pop up since the total quantity may not be lower than or equal to the executed quantity. Thus, in the FIX case, users are always required to inform the total order quantity desired.

3.2.2 Cancellation or Modification of Previously Modified Order – API and FIX

In the case of both API and FIX, to cancel or modify a previously modified order, it is always necessary to inform the last order control number.

3.3 Restrictions

- The order's user code modification, even at auctions, will not be allowed for families of ports #300 through 309 and #500 through 509.
- Placing an order at opening price during auctions and calls will only be allowed for families of ports #400, 500 and 510.
- The user code field should not exceed 8 digits, including the check digit, without a hyphen. Compliance with these restrictions should be observed in sending, cancelling and modifying orders.

3.4 References

- FIX Protocol

http://www.fixprotocol.org

- QuickFIX - Engine FIX Open-Source
  http://www.quickfixengine.org

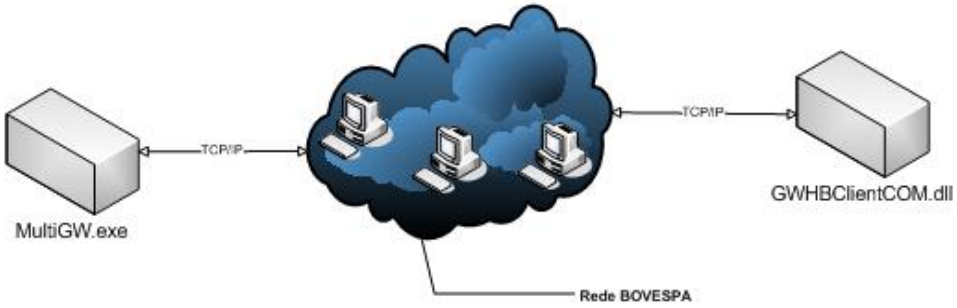
II – Procedures

1 –API PROTOCOL

1.1 Connection

The connection between client and MultiGateway server via API HomeBroker takes place in the following manner.

The client application, using the GWHBClientCOM.dll component, initiates the connection with the server (placing the TCP/IP socket in listening mode) and stands by for the connection.



The MultiGateway server, in turn, performs a number of successive attempts to connect with the client, at time intervals defined by the system administrator on the cluster resources setup screens. There, both the number of attempts and the interval between attempts are set up.

Once it is running, the MultiGateway server always tries to connect with its respective client which was previously set up on the administrator screen, in order to attempt to keep this connection active.

As soon as the server's connection with the client is active, message exchange begins. The client submits an authentication request message and the server responds by either allowing or not allowing a new message to be sent. A detailed description of this mechanism is provided below.

For optimal data distribution, the system was divided into two layers: **session layer and application layer.** These are, in actuality, two conceptual layers with each layer in charge of specific activities.

1.2 Session Layer

The session layer is responsible for the physical communication between client and server. It is in charge of opening the connection between the sockets, message sequencing and retransmission. It contains specific messages to this end, which are handled by this layer only and are not sent to the application layer.

The messages exchanged between server and client include a header and a body. The header is made up of fixed fields which are identical for all messages. The fields comprising the body are specific for each message. The header structure is as follows:

Field name	Format	Description
Wsize	Numerical	Stores the size in Bytes of the message to be sent.
bytLayer	Byte	Informs the destination layer of the message. 1 – session layer, 2 – application layer.
bytType	Byte	Informs the message type. Ex: 1 – authentication message, 2 –

		authentication message OK.
dwSeq	Numerical	Message sequencial number. Will be detailed in the Message Sequencing item.
dwAck	Numerical	Message reception sequential number. Will be detailed in the Message Buffer for subsequent resend.

1.2.1 Message Sequencing

Message sequencing is performed by using the dwSeq and dwAck fields on the message header. The dwSeq field informs the sequential number of a sent message, starting from zero. The dwAck field informs the number of the last message received, processed and sent by the other leg.

Every time a connection between server and client is established, both client and server reset their sequencing values. At each new message sent, the dwSeq value is increased by 1. When one of the legs receives a message it immediately updates its dwAck value to match the value received in the dwSeq field.

In this manner, message sequencing takes place while ensuring that the messages are processed in the order they were sent.

1.2.2 Resend Message Buffer

In order to ensure that messages are sent, both server and client store all messages sent to the other leg in a buffer. This is done so as to address those cases where it becomes necessary to resend previously sent messages, on account of network glitches, for example.

By reviewing the dwAck field each time a new message is received, it is possible to know what was the last message processed by the other leg. Thus, based on such information, all messages whose sequence number is equal to or lower than the dwAck can be flushed from the buffer memory.

1.2.3 Resend Mechanism

Every time a message is received, the dwSeq field is matched against an expected value (dwSeq of the previous message added to 1). Should this value be higher than expected, it means that messages have not been received. Thus, it is necessary to request the other leg to resend the messages not yet received so as to reestablish the correct sequence.

1.2.4 Active Connection Check

The HeartBeat - ImAlive mechanism is used to verify that the connection between client and server remains active even after a time interval has elapsed without any messages being received.

After a period of inactivity which is set up by the system administrator, the server sends a HeartBeat message to the client. Upon receiving this type of message, the client sends another ImAlive type message to the server.

In the event that the server does not receive an ImAlive type message after a certain period of time has elapsed, connection with client is closed. In like manner, if the client fails to receive a HeartBeat type message after a time interval, it will close the connection with the server. After the connection has been closed, the Gateway server once again attempts to establish a new connection with the client.

1.2.5 Session Layer Messages

1.2.5.1 Authentication Request Message

Type: Authentication Request.

Purpose: Send authentication data to the MultiGateway server.

This is the first message to be routed after a physical connection with the server has been established.

Structure: sMsgAuth

Source: Client

Destination: Server

```
struct sMsgAuth
{
sClientMsgHeader sHeader;
union
{
        struct
        {
                char cIdCorretora[5];
                char cIdVersao   [5];

                char cOSPlatformId   [6];
                char cOSMajorVersion [6];
                char cOSMinorVersion [6];
                char cOSBuildNumber  [6];
                char cOSCSDVersion   [15];
                char cOSServicePackMajor[6];
                char cOSServicePackMinor[6];

                char cProcessorArchitecture[6];
                char cProcessorType        [6];
                char cProcessorLevel       [6];
                char cProcessorRevision    [6];
                char cProcessorSpeed       [6];
                char cMemory               [6];

                char cIP                [15];
                char cPort                [5];
                char cSource;

                }; // 120 Bytes -> 8 (oito) Blocos são necessários

        char cFiller     [8 * BLOCO_CRIPTO];

} sBody;
};
```

1.2.5.2 Authentication Success Message

Type: Successful authentication
Purpose: To authenticate the client to initiate system activities. It informs the number of the last message index handled, the last control number received as well as the inactivity time period to be observed before the client sends an ImAlive type message.
Structure: sMsgAuthOK
Source: Server
Destination: Client

```
struct sMsgAuth
{
        sClientMsgHeader sHeader;
    union
    {
    struct
        {
    DWORD dwLastIndex;              // 4 Bytes
            char  cLastCtrlNum[20];          // 20 Bytes
            DWORD dwInactivityTimeout;    // 4 Bytes
            };   // 28 Bytes -> 2 (dois) Blocos são necessários
    char cFiller     [2*BLOCO_CRIPTO];
            } sBody;
};
```

1.2.5.3 Authentication Error Message

Message: Authentication error
Purpose: To warn client against authentication errors.
Structure: sMsgAuthErr
Source: Server
Destination: Client

```
struct sMsgAuthErr
{
        sClientMsgHeader sHeader;
        union
        {
                struct
                {
                        BYTE bytReason;
                }; // 1 Byte -> only one (1) Block is required

                char cFiller     [BLOCO_CRIPTO];
        } sBody;
};
```

1.2.5.4 Message Resend Request Message

Type: Message resend
Purpose: To request the other leg to resend a given message interval.
Structure: sMsgResync
Source: Server / Client
Destination: Client / Server

```
struct sMsgResync
{
	sClientMsgHeader sHeader;
	union
	{
		struct
		{
			DWORD dwBegin;
			DWORD dwEnd;
		}; // 8 Byte -> only one (1) Block is required

		char cFiller	[BLOCO_CRIPTO];
	} sBody;
	    };
```

1.2.5.5 New Message Sequence Notification Message

Type: Notification of a New Message Sequence
Purpose: To inform the other leg about a new message sequence.
Structure: sMsgGapFill
Source: Server / Client
Destination: Client / Server

```
struct sMsgGapFill
{
	sClientMsgHeader sHeader;
	union
	{
		struct
		{
			DWORD dwNewSeq;
		}; // 4 Byte -> only one (1) Block is required

		char cFiller	[BLOCO_CRIPTO];
	} sBody;
};
```

1.2.5.6 Heartbeat Message

Type: HeartBeat
Purpose: To send to client a message informing that it is still active and expecting an ImAlive type message from client.
Structure: sMsgHeartBeat
Source: Server
Destination: Client

```
struct sMsgHeartBeat
{
	sClientMsgHeader sHeader;
```

```
};
```

1.2.5.7 Heartbeat Response Message

Type: Response to heartbeat
Purpose: To send server a message informing that it is still active to avoid disconnection due to idle timeout.
Structure: sMsgImAlive
Source: Client
Destination: Server

```
struct sMsgImAlive
{
        sClientMsgHeader sHeader;
          };
```

1.3 Application Layer

The application layer is responsible for all issues pertaining to server-client negotiation and runs independently from the session layer. It is in charge of handling application messages and interacting directly with client application. It contains a set of specific layer messages and makes use of the session layer to send and receive messages.

Message sizes are not fixed and are encoded according to a standard defined by BOVESPA. Each message has specific fields depending on their purposes, although the fields listed below are common to all messages:

Field name	Format:	Description
M_strIndex	String	Application message index
M_strMsgTransmissionTime	String	Date/Time message was sent

1.3.1 Index

The index field (m_strIndex) is responsible for application message sequencing. Unlike the session layer, where sequencing is achieved on a per connection basis, the application message sequence is done on a daily basis.

This sequencing allows all application messages to be indexed, stored and made available to the client throughout the day.

1.3.2 Control Number

The Control Number field, found in most messages, conveys information which, although not relevant to the Gateway, functions as an order identifier for client applications. The Gateway server always stores the last Control Number used, which corresponds to the same number sent in the client's authentication confirmation message.

1.3.3 Resend Sequence Control

Every time a message is sent to the client, the server expects an Ack type message informing that the message was properly processed by the client. This message comprises information to help the server control how application messages are sent and resent.

With each authentication, the Gateway server sends its client the last index and the last Control Number it has processed. On this occasion, the server reviews the Ack messages received in the last connections, finds out which messages were actually processed by the client and then, if necessary, resends the lost messages.

The client application can also randomly request that old messages be resent using the Recall Messages message, where the indices of the first and last messages to be resent are defined.

1.3.4 Application Layer Messages

1.3.4.1 New Order Registration Request Message

Type: New order
Purpose: To send a new order to the trading system
Class: CAppMsgNewOrder
Source: Client
Destination: Server

Field	Description
strCtrlNumber	ControlNumber, number that identifies the order for the client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 - limit price, 3 – opening price. **See Chapter I, item 3.3**
eovValidity	Validity type: 0 – today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
m_strDisplayQuantity	Quantity to be displayed – Order with disclosed quantity.

1.3.4.2 Order Cancellation Request Message

Type: Order cancellation
Purpose: To send an order cancellation request to the trading system.
Class: CAppMsgCancel
Source: Client
Destination: Server

Field	Description
strCtrlNumber	ControlNumber, number that identifies the order for the client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price. **See Chapter I, item 3.3.**
eovValidity	Validity type: 0 – today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
strExchangeNumber	Number that identifies an order on Gateway. **See Chapter I, item 3.2.2.**

1.3.4.3 Order Rerequest Message

Type: Rerequest of previous orders
Purpose: To request that previously sent orders be resent by server
Class: CAppMsgRecallMsgs
Source: Client
Destination: Server

Field	Description
strBegin	Index of first message to be resent.
strEnd	Index of last message to be resent.

1.3.4.4 Limit Query Message

Type: Limit query
Purpose: To query the limits of a given customer
Class: CAppMsgQueryLimits
Source: Client
Destination: Server

Field	Description
strCtrlNumber	Control number that identifies a request.
strUserID	User ID code.

1.3.4.5 Last Index Query Message

Type: Query the last index.
Purpose: To request Gateway server for the last index used
Class: CAppMsgQueryIndex
Source: Client
Destination: Server

Field	Description
strCtrlNumber	Control number to identify request.
strUserID	User ID code.

1.3.4.6 Order Book Query Message

Type: Query the Order Book
Purpose: To request Gateway server for differential information about a specific message interval
Class: CAppMsgQueryOrderBook
Source: Client
Destination: Server

Field	Description
eobType	Query type 0 – By index, 1 – last ones
strBegin	Index of first message
strEnd	Index of last message.

1.3.4.7 Ack Message

Type: Ack
Purpose: To inform server that the received message was properly processed. It is always sent after an application message received by client has been handled.
Class: CAppMsgAck
Source: Client
Destination: Server

Field	Description
strIndex	Message index.
strMsgTransmissionTime	Message handling time.
strIndexAck	Index of message received by client.
bytOriginalMsgType	Type of Message received by client.
strOriginalMsgReceptionTime	Message reception time.
strCtrlNumber	Control number to identify request.
strExchangeNumber	Transaction identification number on Gateway server.
strStock	Stock symbol used in transaction.

1.3.4.8 Order Registration Confirmation Response Message

Type: Order entry response
Purpose: Message sent after confirmation that an order has been entered into the trading system.
Class: CAppMsgOrderResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price.
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
strOrderTime	Date when order was registered on trading system.
strExchangeNumber	Order identification code on trading system.
m_strOrderTime	Date when order was registered on trading system.
m_strTotalExecutedQuantity	Original order's total executed quantity.

1.3.4.9 Order Cancellation Confirmation Response Message

Type: Order cancellation response
Purpose: Message sent after confirmation that an order was cancelled on the trading system.
Class: CAppMsgCancelResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Symbol/code of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price.
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
strCancelTime	Date when order was cancelled on trading system.
strExchangeNumber	Order identification code on trading system.
eccCondition	Cancellation condition: 0 – Partial, 1 – Total.
strCancelledQuantity	Cancelled quantity.

1.3.4.10 Error Event Response Message

Type: Error event response
Purpose: Message sent after an error has been detected when handling a request.
Class: CAppMsgOrderResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price.
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
strErrorTime	Error event date.
strOrderStatus	Order status.
strErrorSeverity	Error severity.
strErrorNumber	Error code number.
strErrorMsg	Error message.

1.3.4.11 Order Execution Response Message

Type: Order execution response
Purpose: Message sent after order was executed on the trading system.
Class: CAppMsgOrderResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3– opening price.
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
strExecutionTime	Date and time of order execution.
strExchangeNumber	Identification code for each transaction on trading system.
etcCondition	Execution condition: 0 – Partial, 1 – Total.
strTradeQuantity	Executed trade quantity.
strTotalExecutedQuantity	Original order's executed quantity.
strRemainingQuantity	Remaining quantity.
strExecutedPrice	Executed price.
strCounterpartBroker	Identification of counterpart broker.
strTradeId	Trade identification.

1.3.4.12 Index Query Response Message

Type: Index query response.
Purpose: Message sent after index query request.
Class: CAppMsgIndexResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strLastIndex	Last message index used.
strLastCtrlNumber	Last Control Number used by client.

1.3.4.13 Limit Query Response Message

Type: Limit query response.
Purpose: Message sent after limit query request.
Class: CAppMsgLimitResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strUserID	User code.
strTotalLimit	Customer limit total.
strConsumedLimit	Consumed limit.
strPeriod	Period.

1.3.4.14 Order Book Query Response Message

Type: Response to Order Book query.
Purpose: Message sent after Order Book query request.
Class: CAppMsgOrderBookResponse
Source: Server
Destination: Client

Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price.
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Order validity date.
strOrderTime	Date when message was processed.
strExchangeNumber	Order identification code on trading system.
strCumulativeQuantity	Cumulative quantity of previously executed order.
strRemainingQuantity	Remaining quantity of order not yet executed.
strAveragePrice	Average execution price.
strOrderBookIndex	Order Book index.
strNumberOfTrades	Number of executed trades.
strStatusOfOrder	Status of order.
strNoMsgsAvailable	Indicates whether sent message was the last Order Book message.
m_strDisplayQuantity	Quantity to be displayed – Order with disclosed quantity.

1.3.4.15 Order Modification Request Message

Type: Order modification
Purpose: Send an order modification request to trading system.
Class: CAppMsgModify
Source: Client
Destination: Server

Field	Description
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code. **See Chapter I, item 3.3.**
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price. **See Chapter I, item 3.3.**
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded. **See Chapter I, item 3.2.1.**
strPrice	Price per share.
strValidityDate	Validity date
strExchangeNumber	Number that identifies an order on Gateway. **See Chapter I, item 3.2.2.**
m_strDisplayQuantity	Quantity to be displayed – Order with disclosed quantity.

1.3.4.16 Order Modification Confirmation Response Message

Type: Order modification response
Purpose: Message sent after confirmation that order has been modified on the trading system.
Class: CAppMsgModifyResponse
Source: Server
Destination: Client

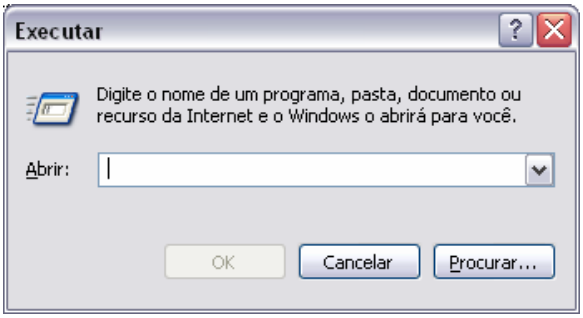
Field	Description
strIndex	Message index.
strCtrlNumber	ControlNumber, number that identifies the order for client application.
strStock	Code/symbol of stock to be traded.
strUserID	Exchange user's code.
eotType	Order type: 0 – buy, 1 – sell.
eocCondition	Condition for order execution: 0 – limit price, 3 – opening price.
eovValidity	Validity type: 0 - today, 1 – until cancellation, 2 – specific date, 3 – all or nothing at all, 4 – execution or cancellation.
strQuantity	Quantity of shares to be traded.
strPrice	Price per share.
strValidityDate	Validity date
m_strExchangeNumber	Order identification code on trading system.
m_strOrderTime	Date when order was registered on trading system.
m_strTotalExecutedQuantity	Original order's total executed quantity.
m_strQtdDispl	Quantity to be displayed – Order with disclosed quantity.

1.4 DLL

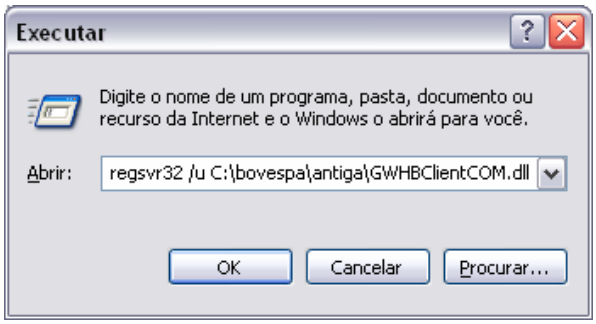
1.4.1 GWHBClientCOM.dll Component Installation

If you already have this component installed on your computer, please follow the steps below to remove your current registration for this component.

- Click on Start → Run


Executar
Digite o nome de um programa, pasta, documento ou recurso da Internet e o Windows o abrirá para você.
Abrir:
OK
Cancelar
Procurar...

- Type the following command line: *regsvr32 /u <path to your current file >GWHBClientCOM.dll*

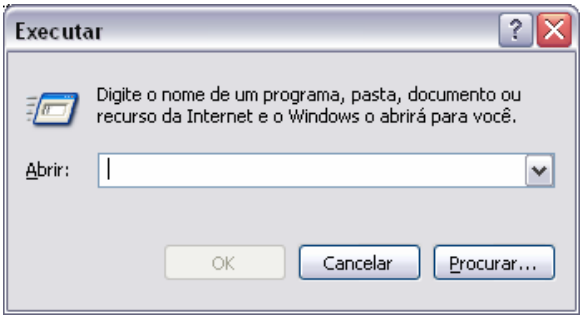


- Click on the "OK" button and the following message will pop up:

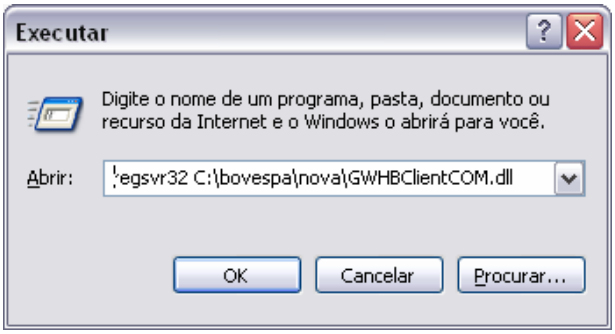
To register the component, follow the steps below:

- Copy the new file GWHBClientCOM.dll to your computer,

- Click on Start → Run


Executar
Digite o nome de um programa, pasta, documento ou recurso da Internet e o Windows o abrirá para você.
Abrir:
OK
Cancelar
Procurar...

- Type the following command line: *regsvr32 <path to your current file >GWHBClientCOM.dll*



- Click on the "OK" button and the following message will pop up:

1.4.2 Basic Use

- Instantiate an HBConnection class object.
- Use the Start method to start the process of connection and authentication on the Gateway Server.
- Await the arrival of AuthenticationOK event to start the send and cancel process.
- Use the object to convey event requests and event reception.
- In the case of an authentication and/or connection error, use Stop method to release resources, followed by Start method to resume the connection process.
- When execution stops, call Stop method to release resources of the HBConnection object.
- In order to use the new functionalities it will be necessary to utilize the HBConnection2 interface, which inherits the methods and properties of HBConnection and implements SendOrder2 (allowing the disclosed quantity to be sent) and ModifyOrder.
- Whenever an OrderBookResponse event is used, the resulting event will share the OrderBookResponse interface. To access the "strDisplayQuantity" property, it is necessary to access the OrderBookResponse2 interface.

Visual Basic -> this is a streamlined conversion that can be performed by creating a new tipoHBOrderBookResponse2 object, for example.

Dim objOrderBookResponse2 As HBOrderBookResponse2
Set objOrderBookResponse = objOrderBookResponse1

Visual C++ -> the HBOrderBookResponse2 interface can be accessed through QueryInterface.

1.4.3 Component Classes

1.4.3.1 HBConnection

Purpose:
Responsible for Gateway connection maintenance allowing requests to be sent and responses to be received in the form of events. It should be instantiated at the start of the application and remain instantiated while the application is running so as to allow events to occur, while the application is also used for Gateway requests.

Properties:
LngStatus As Long

Purpose:
Gateway connection status.

Possible Values:
0 – No connection with server.
1 – Awaiting connection with Gateway server.
2 – Connected to server.
-1 – Connection error.

Methods:
SendOrder (hbOrderInfo As HBOrderInfo)

Purpose:
Send orders to the trading system.

Parameters:
hbOrderInfo – object with order details.

Purpose:
Starts the broker connection process.

Parameters:
LngPort - connection port with Gateway.
IntCorretora - Broker identification.
LngVersao -Client x Gateway protocol version number. Currently, it should be set to 4.

Stop ( )

Purpose:
Terminates connection with Gateway.

CancelOrder (hbCancelOrder As HBCancelOrder)

Purpose:
Cancels an order previously sent to the trading system.

Parameters:
HbCancelOrder – object with details of order to be cancelled.

QueryIndex ( )

Purpose:
Requests that the Gateway provide the last index sent and the last control number received.

QueryLimits(hbQueryLimits As HBQueryLimits)
Purpose:
This function has been deprecated and should not be used.
Requests information about available balance for use on the system.
Parameters:
HbQueryLimits – object with details of a limit query request.

RecallMsgs(hbRecallMsgs As HBRecallMsgs)
Purpose:
Requests message recall.
Parameters:
HbRecallMsgs – object with details about messages required to be resent.

Events:
AuthenticationError(lngError As Long)

Purpose:
Triggered when an authentication error occurs.
Parameters:
LngError – error code.

ConnectionError(lngError As Long)
Purpose:
Triggered when an error occurs in the Gateway Server connection.
Parameters:
LngError - error code.

ConnectionOK()
Purpose:
Triggered when the Gateway connection is successful.

AuthenticationOK(strLastIndex As String, strLastCtrlNum As String)
Purpose:
Triggered when Gateway authentication is successful.
Parameters:
StrLastIndex – index of last message sent.
StrLastCtrlNum – last Control Number sent.

ConnectionClosed()
Purpose:
Triggered when the Gateway Server connection is closed.

OrderResponse(hbOrderResponse As IHBOrderResponse)
Purpose:
Triggered when the Gateway Server sends a response about an order previously sent through the SendOrder() method.
Parameters:
HbOrderResponse – object comprising details of a sent order.

CancelResponse(hbCancelResponse As IHBCancelResponse)
Purpose:
Triggered to inform that an order has been cancelled, which may have been requested through the CancelOrder method or through other trading system tools (Supervisor and/or Mega Control).
Parameters:
HbCancelResponse – object comprising details about the cancellation of an order.

ErrorResponse(hbErrorResponse As IHBErrorResponse)
Purpose:
Triggered in response to a request handling error (Order, Cancellation or Limit Query).
Parameters:
HbErrorResponse – object comprising error details.

IndexResponse(hbIndexResponse As IHBIndexResponse)
Purpose:
Response to an information query about the last indices sent and the control number processed by the Gateway.
Parameters:
HbIndexResponse – object comprising the last index sent and also the last control number processed by the Gateway.

LimitResponse(hbLimitResponse As IHBLimitResponse)
Purpose:
Response to a limit query.
Parameters:
HbLimitResponse – object comprising information about a user's available balance.

TradeResponse(hbTradeResponse As IHBTradeResponse)
Purpose:
Event triggered to inform that a trade has been executed.
Parameters:
HbTradeResponse – object comprising details about the execution of a trade.

TradeCancelResponse(hbTradeCancelResponse As IHBTradeResponse)
Purpose:
Event triggered to inform that a trade has been cancelled.
Parameters:
HbTradeCancelResponse – object comprising details of a trade cancellation.

1.4.3.2 HBConnection2
Purpose:
Interface based on the HBConnection interface aimed at creating new methods and ensuring compatibility.

Methods:

SendOrder2 (hbNewOrder As HBNewOrder)
Purpose:
To send orders to the trading system.
Parameters:
hbNewOrder – object with order details.

ModifyOrder (hbModifyOrder As HBModifyOrder)
Purpose:
To send order modifications to the trading system.
Parameters:
hbModifyOrder – object with order details.

1.4.3.3 HBOrderInfo

Purpose:
Represents the features of an order. It is used in requests and events to define/inform the features of an order.

Properties:

strStock As String
Purpose:
Stock / asset code/symbol.

strCtrlNumber As String
Purpose:
To identify an order or cancellation (created by the client application) – it should identify one request only.

strUserID As String
Purpose:
Customer code associated with an order (check digit).
Note: The user code field should not exceed 8 digits, including the check digit, without a hyphen.

eotType As HBOrderType (enum)
Purpose:
Order type.
Possible Values:
0 – Buy
1 – Sell

strQuantity As String
Purpose:
Quantity of shares.

eocCondition As HBOrderCondition
Purpose:
Order condition (price type).
Possible Values:
0 – Limit price
1 – Market price
2 – Best price
3 – Opening price
4 – Any price
5 – Best sell
6 – Best buy
7 – Last price
8 – Sell price
9 – Buy Price

strPrice As String
Purpose:
Share price. It should be completed whenever the condition (price type) equals "0 – Limit Price". It uses a dot ('.') as decimal separator and supports up to 6 decimal places. Thousand separators are not used. It can comprise up to 9 significant digits.

eovValidity As HBOrderValidity
Purpose:
Order validity type.
Possible Values:
0 - Today
1 – Until cancellation
2 – Specific date
3 – All or nothing at all
4 – Execute or cancel

strValidityDate as String
Purpose:
Order validity date in the following format: YYYYMMDD. It should be completed whenever validity type equals "2 – Specific Date".

1.4.3.4 HBCancelOrder
Purpose:
Relevant information for order cancellation. Used for cancellation requests.

Properties:
hbOrderInfo As HBOrderInfo
Purpose:
Details of order to be cancelled.

StrExchangeNumber As String
Purpose:
Exchange Number of order to be cancelled.

1.4.3.5 HBQueryLimits
Purpose:
Relevant information about limit queries.

Properties:

strCtrlNumber As String
Purpose:
Request identifier.

StrUserID As String
Purpose:
User associated with a query.

Note: The user code field should not exceed 8 digits, including the check digitwithout a hyphen.

1.4.3.6 HBRecallMsgs

Purpose:
Relevant information for message recalls.

Properties:

strBegin As String
Index of the initial message whose resend is being requested.

strEnd As String
Index of the final message whose resend is being requested.

1.4.3.7 HBOrderResponse

Purpose:
Information about the acceptance of an order by the trading system.

Properties:

StrExchangeNumber As String
Purpose:
Number attributed to an order by the trading system.

strOrderTime As String
Purpose:
Time when an order was registered on the system; use YYYYMMDDHHMMSS format.

strIndex As String
Purpose:
Daily sequential index of sent message. In the case of a sequence breakdown, the application should request that messages be resent.

hbOrderInfo As HBOrderInfo
Purpose:
Object comprising details of an accepted order.

1.4.3.8 HBCancelResponse

Purpose:
Information about the cancellation of an order by the trading system.

Properties:

StrExchangeNumber As String
Purpose:
Number attributed to a cancelled order by the trading system.

strIndex As String
Purpose:
Daily sequential index of the sent message. In the case of a sequence breakdown, the application should request that messages be resent.

hbOrderInfo As HBOrderInfo
Purpose:
Details of cancelled order.

strCancelledQuantity As String
Purpose:
Cancelled quantity.

hbCancelCondition As HBCancelCondition
Purpose:
Cancellation condition
Possible Values:
0 - Partial
1 – Total

strCancelTime As String
Purpose:
Time when order cancellation was executed; use YYYYMMDDHHMMSS format.

1.4.3.9 HBErrorResponse

Purpose:
Information about error event.

Properties:

hbOrderInfo As HBOrderInfo
Purpose:
Details of an order which caused an error, or details of the order associated with the cancellation that caused the error.

strErrorMsg As String
Purpose:
Textual error message; According to a Gateway query table.
If user wishes to display messages in different languages, the ErrorNumber property should be used to translate messages into the desired language.

strErrorNumber As String
Purpose:
Numerical error code.

strErrorSeverity As String
Purpose:
Error severity.
Possible Values:
0 – Gateway
1 – SLE
2 – MEGA BOLSA

strErrorTime As String
Purpose:
Error event time.

strIndex As String
Purpose:
Daily sequential index of sent message. In the case of a sequence breakdown, the application should request that messages be resent.

strOrderStatus As String
Purpose:
Order status on the trading system.
Possible Values:
5 – Order on hold. When this situation occurs, the order remains on the system awaiting either acceptance or rejection; as a result, the order is not disregarded. If the order is accepted, an OrderResponse event will confirm the order; otherwise, a new error or cancellation event will invalidate the order.
Other Values – Order Rejected (not registered).

1.4.3.10 HBIndexResponse
Purpose:
Information regarding a response to an index query.

Properties:

strLastCtrlNumber As String
Purpose:
Last Control Number processed by the Gateway.

strLastIndex As String
Purpose:
Index of last message sent by the Gateway.

1.4.3.11 HBLimitResponse
Purpose:
Information about user's trading limit.

Properties:

strConsumedLimit As String
Purpose:
Consumed limit value.

strCtrlNumber As String
Purpose:
Request identifier (query).

strIndex As String
Purpose:
Daily sequential index of a sent message. In the case of a sequence breakdown, the application should request that messages be resent.

strPeriod As String
Purpose:
Number of days before limit is reset.

strTotalLimit As String
Purpose:
User's total limit.

strUserID As String
Purpose:
Code of user whose limit has been queried.
Note: The user code field should not exceed 8 digits, including the check digit without a hyphen.

1.4.3.12 HBTradeResponse

Purpose:
Information about trade execution or cancellation.

Properties:

hbTradeCondition As HBTradeCondition
Purpose:
Trade condition.
Possible Values:
0 – Partial execution
1 – Total execution

hbOrderInfo As HBOrderInfo
Purpose:
Order information details.

strAveragePrice As String
Purpose:
Average price of all executions associated with an order.

strCounterpartBroker As String
Purpose:
Counterpart broker.

strExchangeNumber As String
Purpose:
Order exchange number.

strExecutedPrice As String
Purpose:
Trade executed price.

strExecutionTime As String
Purpose:
Trade execution time.

strIndex As String
Purpose:
Daily sequential index of sent message. In the case of a sequence breakdown, the application should request that messages be resent.

strRemainingQuantity As String
Purpose:
Remaining quantity.

strTotalExecutedQuantity As String
Purpose:
Total executed quantity (for all executions associated with an order).

strTradeQuantity As String
Purpose:
Trade quantity executed.

1.4.3.13 HBModifyResponse

Purpose:
Information about acceptance of an order modification.

Properties:

strExchangeNumber As String
Purpose:
Number attributed to an order by the trading system.

strOrderTime As String
Purpose:
Time when an order was registered on the system in YYYYMMDDHHMMSS format.

strIndex As String
Purpose:
Daily sequential index of sent message. In the case of a sequence breakdown, the application should request that messages be resent.

hbOrderInfo As HBOrderInfo
Purpose:
Object comprising details of an accepted order.

strDisplayQuantity As String
Purpose:
Quantity to be displayed for orders with disclosed quantity.

1.4.3.14 HBNewOrder

Purpose:
Represents the features of a new order. Used for send order requests.

Properties:

hbOrderInfo As HBOrderInfo
Purpose:
Order information.

strDisplayQuantity As String
Purpose:
Quantity to be displayed for orders with disclosed quantity.

1.4.3.15 HBModifyOrder

Purpose:
Represents the features of a new order. Used for send order requests.

Properties:

hbOrderInfo As HBOrderInfo
Purpose:
Order information.

strDisplayQuantity As String
Purpose:
Quantity to be displayed for orders with disclosed quantity.

strExchangeNumber As String
Purpose:
Number attributed to an order by the trading system.

1.4.3.16 HBOrderBookResponse

Purpose:
Information about the acceptance of an order by the trading system. Return information to customer with regard to user's Order Book.

Properties:

strIndex As String
Purpose:
Daily sequential index of sent message. In the case of a sequence breakdown, the application should request that messages be resent.

strOrderTime As String
Purpose:
Time when order was registered on the system, in YYYYMMDDHHMMSS format.

strExchangeNumber As String
Purpose:
Order Exchange Number.
strCumulativeQuantity As String
Purpose:
Cumulative quantity of executed order.

strRemainingQuantity As String
Purpose:
Remaining quantity.

strAveragePrice As String
Purpose:
Average execution price.

strNumberOfTrades As String
Purpose:
Number of trade executions.

strStatusOfOrder As String
Purpose:
Order status.

blnNoMessagesAvailable As String
Purpose:
Indicates whether or not orders are available.

strOrderBookIndex As String
Purpose:
Order entry index in Order Book.

hbOrderInfo As HBOrderInfo
Purpose:
Object comprising detailed information about an accepted order.

1.4.3.17 HBOrderBookResponse2

Purpose:
Interface based on the HBConnection interface aimed at creating new methods and ensuring compatibility.

Properties:

strDisplayQuantity As String
Purpose:
Quantity to be displayed for orders with disclosed quantity.

1.5 LIB

1.5.1 LibGnr.a Component Installation

To register the component, follow the steps below:
- Copy the new LibGnr.a file in binary mode and the GnrCli.h file in ASC mode to your computer.
- Linkedit along with client application.

Linkedition example:
If AplCli.c is the client application, the following line should be added:
#include Gnrcli.h in the program text
and to linkedit, use the following command:
cc AplCli.c libGnr.a -o AplCli

1.5.2 Basic Use

The LIB (Library, i.e., a Unix library added to the client application to access functions) is provided by the client application developers for linkedition. It consists of functions called by the client application. These functions start up listening for a connection and authentication on the Gateway, and once these are accomplished, methods are sent to the Gateway. The LIB has two "fronts": one that awaits and sends messages to/from the Gateway and one that awaits the methods and sends events to the client application. The process that communicates with the client application is called "pai" (Parent). The process of sending and receiving messages by the Gateway through the socket is called "filho" (Son). Upon receiving a method from the client the parent process writes to a pipe and is read by the son process, which also monitors – via select – the Gateway socket. Upon receiving a message from the Gateway addressed to the parent, the son process sends a signal (SIGUSR1 or SIGUSR2) to the parent, thereby making it aware that there is a message for it on the pipe. The parent then processes the message and generates an event to the client application through a callback function, which was informed by the client application when the Start method was executed (see 1.5.3.3).

The LIB is associated with the session layer and application layer concepts. The session layer only handles message exchanges and it contains mechanisms to ensure that the connection reception/transmission of messages are maintained on a continuous and sequential manner. Every message has a header to provide the other leg with information such as the sequential number of the current message and the last sequential number received by one of the legs. Without the awareness of the client application, both the Gateway and the LIB can request messages to be resent if a sequence breakdown has been noted by one of the legs. Besides, they exchange HeatBeat messages (sent by the Gateway) and ImAlive messages (responded by the LIB). TimeOut is yet another control: once Gateway accepts authentication, it sends an AUTHOK type message comprising a TimeOut value, among other information. This value indicates the maximum time period allowed for the LIB to receive no messages from the Gateway. To check this timing, the LIB makes use of a **SetTimer** function, which issues an alarm, and as soon as the set time has elapsed, triggers the **TimeOut** function. The TimeOut function simply sets to TRUE the

SharedMemory's ucFim variable to break the monitoring loop. A **ResetTimer** is executed for each message received from the Gateway.

Message reception/transmission by the application layer is ensured by an ACK sent to the Gateway when the client application's event handling is completed.
The application layer therefore remains independent from the session. Should the client application "stall" an event (i.e., delays in handling it) the session layer, nevertheless, continues its communication with Gateway by exchanging HeartBeat/ImAlive messages or receiving exchange responses. Should the client application "hang" and Gateway messages begin to stack up, the LIB chooses to terminate the Gateway connection. Since the Gateway maintains a list of unhandled messages (in other words, messages that did not send an ACK), the Gateway resends all past pending messages from the next/new connection.
To close the whole process, the client application executes the **Stop** method, which simply sets to TRUE the SharedMemory's ucFim variable to flag the end of the monitoring loop.

1.5.3 Functions

1.5.3.1 GetStatus

Syntax:

unsigned int **GetStatus** (void);

Returns the connection status with MEGA BOLSA, where the following values are possible:

- **CLOSED**, when the UNIX client is created but is not yet prepared to connect with MEGA BOLSA, or immediately after handling an error that caused a connection failure.
- **WAITING**, client initialized and ready to connect with MEGA BOLSA.
- **CONNECTED**, connection with MEGA BOLSA is successful; awaiting authentication.
- **AUTHENTICATED**, after validating its credentials (**BrokerId/IP/Version**) on the Gateway, it is ready to send and receive messages.
- **ERROR**, immediately after a communication error which has not yet been handled.

1.5.3.2 GetLastCtrlNumber

Returns the sequential number of the last message successfully sent by the Home Broker application to MEGA BOLSA, thereby allowing the application to have message retrieval control.

Syntax:

char * **GetLastCtrlNumber** (void);

1.5.3.3 Start

Responsible for allocating the necessary resources; it starts by waiting for a connection with MEGA BOLSA, then changing the status from CLOSED to WAITING. It accepts the following events as parameters:

- **iPort:** Represents the socket port used by Gateway to attempt a connection.
- **uiBrokerId:** Broker identification code.
- **pfnEvents:** Pointer to the (callback) function to handle an occurred event.

Syntax:

```
#typedef void (* PFN_EVENT_CALLBACK)( unsigned char ucType,
                                      void * lpvParam);
int     Start  (int iPort, unsigned int uiBrokerId,
                PFN_EVENT_CALLBACK pfnEvents );
```

Event may occur due to messages sent to MEGA BOLSA in response to the **Start** function. These events are:

- **ConnectionAccepted():** Occurs in response to a connection with MEGA BOLSA, when the Status is changed from WAITING to CONNECTED.
- **ConnectionError():** Occurs in response to a communication error with MEGA BOLSA so that a user is warned about a status transition to ERROR and subsequently to CLOSED.
- **AuthenticationOK():** Occurs when the broker application is authenticated on the Remote Trading Gateway, and the Status changes from CONNECTED to AUTHENTICATED.
- **AuthenticationError():** Occurs when the broker application fails to authenticate, and the Status changes from CONNECTED to CLOSED.

Note: The connection event *ConnectionClosed()* occurs when the Remote Trading Gateway is closed. In like manner, as in the case of a *ConnectionError()*, after the occurrence of a *ConnectionClosed()*, the **Start** function should be used to establish a new connection, thus imparting greater control to the broker application. Under no circumstances will the reconnection ever occur automatically.

1.5.3.4 Stop

Responsible for closing the connection with MEGA BOLSA to release the allocated resources and for changing the Status to CLOSED.

Syntax:

long **Stop**();

1.5.3.5 SendOrder

Responsible for sending Buy and Sell Orders, the **SendOrder()** function accepts as parameters the sequential number of the transaction, the broker's client number and a TypeOrder type structure.

Syntax:

long **SendOrder**(char * pcCtrlNumber, char * pcUserID,
*TypeOrder *pOrder);*

Where:

pcCtrlNumber should be the pointer to a broker's internal sequential number with ten (10) digits, ending in '\0'.

The sequential number should observe the MMDDSSSSSS format, where MM and DD stand respectively for the month and day the order is being executed and SSSSSS corresponds to the sequential number on the order day.

The GNROrder structure contains information about the sent order, observing the following format:

```
typedef struct {
    int     iOrderType      ;
    char    pcStock[13]     ;
    char    acQuantity[13]  ;
    char    pcPrice[11]     ;
    int     iCondition      ;
    int     iValidity       ;
    char    pcValidityDate[9] ;
```

```
    char    acDisclosedQty[13] ; /* field entry as of version 1.3.0.0 */
} TypeOrder;
```

Where

- **OrderType** corresponds to the sent order type, either buy or sell, according to the GNROrderType numbered type.

```
Enum GNROrderType {
        otBuy = 0,
        otSell = 1
    };
```

- **Stock:** Stock and market code/symbol (according to BOVESPA's standard).
- **Quantity:** Quantity ordered.
- **acDisclosedQty:** Disclosed quantity.
- **Price:** Order price (construed in accordance with the trading condition).
- **Condition:** Trading condition (limit price, opening price, best price, market price), according to the GNROrderCondition numbered type:

```
Enum GNROrderCondition {
        cndLimitPrice = 0,
        cndMarketPrice = 1,
        cndBestPrice = 2,
        cndOpenPrice = 3,
        cndAnyPrice = 4,
        cndBestSell = 5,
        cndBestBuy = 6,
        cndLastPrice = 7,
        cndSellPrice = 8,
        cndBuyPrice = 9
        };
```

- **Validity:** Order validity (for the day, until cancellation – 30 days maximum, all or nothing at all, executes or cancels, until a designated date), according to the GNROrderValidity numbered type:

```
Enum GNROrderValidity {
        vldToday = 0,
        vldUntilCancel = 1,
        vldDate = 2,
        vldAllOrNothing = 3,
        vldExecuteOrCancel = 4
    };
```

- **Validity Date:** Limit date of order validity, if such validity extends to a specific date.
- **IgnoreLimit:** Enables orders to be sent with values above the daily trading limit or orders whose value – added to a client's current balance – also exceeds the aforementioned limit.

Note: Upon receiving an order whose value exceeds the daily trading limit, the Gateway will adopt the following procedure:

•If the order has an enabled IgnoreLimit field, an "Error Response" will be issued to indicate improper use of this field.
•If the order has a disabled IgnoreLimit field, a "Limit Response" will be issued, i.e. the adopted procedure will be normal and the order will be stored pending future authorization. In order to attain authorization, the client should resend the order (using the same Control Number) with the IgnoreLimit enabled.

•When the previously rejected order is resent with the IgnoreLimit field enabled, the Gateway will accept it and add its value to the client's current balance.

An order whose value does not exceed the daily trading limit but has the IgnoreLimit field enabled will cause the Gateway to issue an **Error Response()** event indicating improper use of this field.

If the order is accepted, an **OrderResponse()** event will be received with information about the order's registration number on MEGA BOLSA, called an Exchange Number. Otherwise, an **ErrorResponse()** event will be received.

1.5.3.6 CancelOrder

Responsible for cancelling an order, the **CancelOrder()** function accepts the following parameters. The sequential number of the transaction, the broker's client number, a TypeOrder type structure and the Exchange Number of the order, received during the **OrderResponse()** event.

Syntax:

```
long CancelOrder(   char * pcCtrlNumber, char * pcUserID,
                    TypeOrder *pOrder, char* pcExchNumber
);
```

Where pcCtrlNumber should be the pointer to the broker's internal sequential number, with ten (10) digits, ending in '\0'; the pcExchangeNumber should be the pointer to the MEGA BOLSA order number, with eleven (11) digits, ending in '\0' and pOrder follows the TypeOrder structure specified under item 1.5.3.5 – SendOrder.

MEGA BOLSA's response to the execution of this function will generate a **CancelResponse()** event.

1.5.3.7 QueryLimits

This function has been deprecated and should not be used.

Responsible for querying a client's balance during a given period, the **QueryLimits()** function accepts two parameters. The sequential number of the transaction and the user who wishes to query the trading limits. The Gateway response to this query will generate a **LimitResponse()** event.

Syntax:

```
long QueryLimits (char * cCtrlNumber, char * pcUserID );
```

Where pcCtrlNumber should be the pointer to a broker's internal sequential number with ten (10) digits, ending in '\0' and pcUserId should be the pointer to a user code with six (6) digits, ending in '\0'.

1.5.3.8 QueryIndex

Responsible for querying the index of the last MEGA BOLSA message returned by the Gateway, as well as the Control Number of the last message successfully sent by the broker application. The Gateway response to this query will generate a **LimitResponse()** event.

Syntax:

long **QueryIndex** (void);

1.5.3.9 RecallMessages

Responsible for requesting the recall of the last response messages sent by MEGA BOLSA. The **RecallMessages()** function accepts the following parameters: The index of a message that enables the recall and discloses the number of messages to be recalled.

Syntax:

long RecallMessages (unsigned int uiIndex, unsigned int uiNumMsgs);

1.5.3.10 QueryOrderBook

Responsible for requesting the transmission of a user's Order Book, i.e., all orders by a given user (300, for example) recorded in a broker's Order Book. The **QueryOrderBook()** function accepts as parameters the request type, the order index to be used for recall and the index of the last order one wishes to recall. The valid values for this type are:
0: When the initial and final values of a query are informed or
1: When one wishes to query the last order only; in this case, all other parameters are disregarded.
Note: the indices pertaining to this query are those found when an order is entered into the Order Book and not those generated by the Gateway for each message sent to Unix LIB.

Syntax:
long QueryOrderBook(unsigned int uiType, unsigned int uiBegin, unsigned int uiEnd);

1.5.3.11 ModifyOrder

Responsible for the modification of an order, the **ModifyOrder()** function accepts as parameters the sequential number of the transaction, the broker's client number, a TypeOrder type structure and the Exchange Number of the order, received during the **OrderResponse()** event.

Syntax:

long **ModifyOrder** (char* pcCtrlNumber, char * pcUserID, TypeOrder *pOrder, char* pcExchNumber)

Where pcCtrlNumber should be the pointer to the broker's internal sequential number, with ten (10) digits, ending in '\0'; the pcExchangeNumber should be the pointer to the MEGA BOLSA order number, with eleven (11) digits, ending in '\0' and pOrder follows the TypeOrder structure specified under item 1.5.3.5 - SendOrder.

MEGA BOLSA's response to the execution of this function will generate a **ModifyResponse**() event.

1.5.4 Events

Events are generated in the form of callback via function pointer in response to closed transactions with the purpose of conveying the results to the Home Broker application. Depending on the transaction type, one or more events can be generated.

In order for the event mechanism to be triggered, the Home Broker should implement a handling function in accordance with the following prototype:

void **TrataEvento** (GNREventsTypes EventId, void * pvParam);

And upon initialization, it should relay its address as a parameter to the **Start()** function.
Example:

Start(TrataEvento**);**

The handling function should identify the event type that was received, interpret its parameters conveyed via a **void *** type variable, and subsequently provide the appropriate handling.
Example:

```
void TrataEvento (GNREventsTypes EventId, void * pvParam){
    switch(EventId){
            ...
            case RSP_ORDER:
                TrataOrderResponse ( (TypeOrderResponse*) pvParam );
                break;
            case RSP_TRADE:
                TrataTradeResponse ( (TypeTradeResponse*) pvParam );
                break;
            ...
    }
}
```

Next, a description of possible events and the parameters received by each event is presented.

1.5.4.1 OrderResponse

It is generated to confirm a **SendOrder()** transaction, accepting as parameter a TypeOrderResponse type structure comprising the following fields:

```
typedef struct {
        char pcCtrlNumber[11];
        char pcUserId[9];
        char pcExchNumber[13];
        char pcOrderTime[15];
        char pcIndex[7];
    } TypeOrderResponse;
```

Where:

- **ControlNumber:** Sequential number of a closed order.
- **UserId:** Client code of the broker requesting the order.
- **ExchangeNumber:** Number of the MEGA BOLSA order (this number should be kept and later sent along with the order in the event of an order cancellation).
- **OrderTime:** Order date/time
- **Index:** Sequential number attributed by Gateway to its response messages.

1.5.4.2 TradeResponse

It occurs when MEGA BOLSA sends a partial or total order execution and receives a TypeTradeResponse type structure comprising the following fields:

```
Typedef struct {
        char pcCtrlNumber[11];
        char pcUserId[9];
        char pcTradeNumber[13];
        char pcExchNumber[13];
        char pcExecutedQuantity[13];
        char pcExecutedTradeQuantity[13];
        char pcRemainingQuantity[13];
        int iCondition;
        char pcPrice[11];
        char pcAveragePrice[11];
        char ExecutedTime[15];
        char pcCounterpartBroker[4];
        char pcIndex[7];
} TypeTradeResponse;
```

Where:

- **ControlNumber:** Sequential number of order being executed.
- **UserId:** Code of client whose order is being executed.
- **TradeNumber:** Trade number on MEGA BOLSA.
- **ExchangeNumber:** Order number which generated a trade.
- **ExecutedQuantity:** Total executed quantity.
- **ExecutedTradeQuantity:** Executed trade Quantity.
- **RemainingQuantity:** Remaining quantity on the order that generated a given trade.
- **Condition:** Execution type, either partial or total.
  - **Price:** Executed price.
  - **AveragePrice:** Average price of all partial executions for a given order.
  - **ExcutedTime:** Trade date/time.
  - **CounterpartBroker:** Number of broker that carried out the trade.
  - **Index:** Sequential number attributed by Gateway to its response messages.

When the **Condition** field value equals **otcTotal**, order execution will have been completed and cannot be cancelled. When the executed trade status equals **otcPartial**, the order may have its non-traded part cancelled.

1.5.4.3 CancelResponse

It occurs when MEGA BOLSA sends a response to a **CancelOrder()** transaction, accepting as parameter a TypeCancelResponse object type structure comprising the following fields:

```
typedef struct {
        char pcCtrlNumber[11];
        char pcUserID[9];
        char pcExchNumber[13];
        char pcCancelQuantity[13];
        int iCondition;
        char pcCancelTime[15];
        char pcIndex[7];
} TypeCancelResponse;
```

Where:

- **ControlNumber:** Sequential number of order cancellation.

- **UserId:** Code of client cancelling an order.
- **ExchangeNumber:** MEGA BOLSA order number.
- **CancelQuantity:** Cancelled quantity.
- **Condition:** The cancellation – either partial or total – according to GNRCancelConditon numbered type:

```
Enums {
                occPartial = 0,
                occTotal = 1
        } GNRCancelCondition;
```

- **CancelTime:** Order cancellation date/time.
- **Index:** Sequential number attributed by Gateway to its response messages.

The **Condition** field returns either **occTotal** or **occPartial**, depending on the order status. If trade execution is already underway, cancellation will be partial; otherwise it will be total. In the event of an attempt to cancel a total execution order, MEGA BOLSA will respond with an **ErrorResponse** event containing the Order Not Found message.

1.5.4.4 LimitResponse

A response is generated to a broker's client limit query, accepting as parameter a TypeLimitResponse type estructure comprising the following fields:

```
typedef struct {
                char pcCtrlNumber[11];
                char pcUserID[9];
                char pcTotalLimit [11];
                char pcConsumedLimit[11];
                int iPeriod;
                char pcIndex[7];
        } TypeLimitResponse;
```

Where:

- **ControlNumber:** Sequential number of limit query.
- **UserId:** Code of client performing the query.
- **TotalLimit:** Maximum limit traded in a given period.
- **ConsumedLimit:** Limit spent by a client in a given period.
- **Period:** Time (in days) remaining before **ConsumedLimit** is reset.
- **Index:** Sequential number attributed by Gateway to its response messages.

Note: The **ConsumedLimit** field contains the summation of order values sent to MEGA BOLSA within a given trading period. This means that a sent order value will be added to the client's balance as soon as it is received by the Gateway, even if its validity is still in force for a few more days.
The cancellation of an order with many days' validity (30, for example) will only cause the client's balance to be written off if the cancellation is made within the same trading period as when the order was sent.

1.5.4.5 ErrorResponse

It occurs when either MEGA BOLSA or the Gateway returns an error, accepting as parameter a TypeErrorResponse type structure comprising the following fields:

```
typedef struct {
```

```
                    char pcCtrlNumber[11];
                    char pcUserID[9];
                    unsigned char ucErrorSeverity;
                    char pcErrorNumber[11];
                    char pcErrorMsg[100];
                    char pcErrorTime[15];
                    char cStatusOfOrder;
                    char pcIndex[7];
            } TypeErrorResponse;
```

Where:

- **ControlNumber:** Sequential number of the transaction that generated an error.
- **UserId:** Broker client's code.
- **ErrorSeverity:** Indicates whether there is any need for external intervention to solve a given problem.
- **ErrorNumber:** Error number.
- **ErrorMessage:** Error description.
- **ErrorTime:** Error event time.
- **Order Status:** Indicates whether an order has been rejected by MEGA BOLSA.
- **Index:** Sequential number attributed by Gateway to its response messages.

1.5.4.6 TradeCancelResponse

It occurs **whenever** MEGA BOLSA sends a partial or total order cancellation message (TradeResponse) and receives a TypeTradeResponse type structure. Under no circumstances will a trade cancellation ever cause a TradeResponse event.

1.5.4.7 IndexResponse

It is generated in response to a query regarding the index of the last MEGA BOLSA message returned by the Gateway, as well as the Control Number of the last message successfully sent by Unix LIB. It accepts as parameter a TypeIndexResponse type structure with the following fields:

```
        typedef struct {
        char pcCtrlNumber[11];
        char pcIndex[7];
    } TypeIndexResponse;
```

Where:

- **ControlNumber:** Sequential number of the last order sent by Unix LIB and successfully received by the Gateway.
- **Index:** Sequential number generated by the Gateway, belonging to the Gateway's last response sent by the Gateway and successfully received by the Unix LIB.

1.5.4.8 OrderBookResponse

It is generated in response to a QueryOrderBook query. It accepts as parameter a TypeOrderBookResponse type structure with the following fields:

```
typedef struct
{
    char pcIndex[7]             ;
```

```
        char pcCtrlNumber[11]        ;
        char pcStock[13]             ;
        char pcUserID[9]             ;
        int  iOrderType              ;
        int  iCondition              ;
        int  iValidity               ;
        char pcQuantity[13]          ;
        char pcPrice[11]             ;
        char pcValidityDate[9]       ;
        char pcOrderTime[15]         ;
        char pcExchNumber[13]        ;
        char pcCumulativeQuantity[13] ;
        char pcRemainingQuantity[13]  ;
        char pcAveragePrice[11]      ;
        char pcOrderBookIndex[7]     ;
        int  iNumberOfTrades         ;
        char pcStatusOfOrder[3]      ;
        char cNoMsgsAvailable        ;
        char acDisclosedQty[13]      ;  /* field input from version 1.3.0.0 */
} TypeOrderBookResponse;
```

Where:

- **pcIndex:** Sequential number attributed by Gateway to its response messages.
- **pcCtrlNumber:** Executed order's identifying number.
- **pcStock:** Stock and market code/symbol (according to the BOVESPA standard).
- **pcUserID:** Client code of the broker requesting the order.
- **iOrderType:** Sent order's type, either buy or sell, according to the GNROrderType numbered type.
- **iCondition:** Trading condition (limit price, opening price, best price, market price), according to the GNROrderCondition numbered type:
- **iValidity:** Order validity (for current day, until cancellation – 30 days maximum, executes or cancels, until a designate date), according to the GNROrderValidity numbered type:
- **pcQuantity:** Quantity ordered.
- **pcPrice:** Order price (construed in accordance with the trading condition).
- **pcValidityDate:** Limit date of order validity, if such validity extends to a specific date.
- **pcOrderTime:** Order date/time
- **pcExchNumber:** MEGA BOLSA order number.
- **pcCumulativeQuantity:** Total executed quantity.
- **pcRemainingQuantity:** Remaining quantity.
- **pcAveragePrice:** Average execution price.
- **pcOrderBookIndex:** Order entry index in Order Book.
- **iNumberOfTrades-:** Number of closed trades.
- **pcStatusOfOrder-:** Order status. It can accept the following values:

'E ' Order waiting (E + SPACE)
'O ' Order accepted (O + SPACE)
'OX' Order partially executed
'OG' Order on hold.
'TX' Order totally executed
'A ' Order cancelled (A + SPACE)

'AX'	Cancelled order partially executed
'R '	Order rejected (R + SPACE)
' '	Order being modified (SPACE + SPACE)
' 'X'	Order partially executed and currently being modified (SPACE + X)
'N '	Order being cancelled (N + SPACE)
'AA'	Order awaiting cancellation
'NX'	Order partially executed and currently being cancelled

- **cNoMsgsAvailable-:** If '0', orders are available; if '1', no orders available.

1.5.4.9 AuthenticationOK

It occurs when a UNIX client is authenticated changing the Status from CONNECTED to AUTHENTICATED. This event accepts as parameter a TypeIndexResponse type structure with the following fields:

```
typedef struct {
        char pcCtrlNumber[11];
        char pcIndex[7];
} TypeAutOkResponse;
```

Where:

- **ControlNumber:** Sequential number of the last order sent by Unix LIB and successfully received by Gateway.
- **Index:** Sequential number generated by MEGA BOLSA, belonging to Gateway's last response, which was sent by Gateway and successfully received by Unix LIB.

1.5.4.10 AuthenticationError

It occurs when a UNIX client's authentication is unsuccessful changing the Status from CONNECTED to CLOSED. This event accepts no parameters and receives a null in pvParam.

1.5.4.11 ConnectionAccepted

It occurs in response to a connection with MEGA BOLSA, when the Status is changed from WAITING to CONNECTED. This event accepts no parameters and receives a null in pvParam.

1.5.4.12 ConnectionError

It occurs in response to a communication error with MEGA BOLSA so that the Home Broker is warned about a status transition to ERROR and subsequently to CLOSED. This event accepts no parameters and receives a null in pvParam.

1.5.4.13 ModifyResponse

It occurs when MEGA BOLSA sends a response to a **ModifyOrder()** transaction, accepting as parameter a TypeModifyResponse object type structure comprising the following fields:

```
typedef struct
{
    char pcCtrlNumber[11]     ;
    char pcUserID[9]          ;
    int  iOrderType           ;
    int  iCondition           ;
    int  iValidity            ;
    char pcQuantity[13]       ;
```

```
        char pcPrice[11]            ;
        char pcValidityDate[9]      ;
        char pcExchNumber[13]       ;
        char pcOrderTime[15]        ;
        char acDisclosedQty[13]     ;
        char acExecutedTradeQuantity[13];
        char pcIndex[7]             ;
} TypeModifResponse;
```

Where:

- **ControlNumber:** Sequential number for an order modification.
- **UserId:** Code of client modifying an order.
- **ExchangeNumber:** MEGA BOLSA order number.
- **Condition:** Refers to the modification type and can be either partial or total
- **Index:** Sequential number attributed by Gateway to its response messages.
- **iOrderType:** Sent order's type, either buy or sell
- **iValidity:** Order validity (for current day, until cancellation – 30 days maximum, executes or cancels, until a designated date)
- **pcQuantity:** Quantity ordered.
- **pcPrice:** Order price (construed in accordance with the trading condition).
- **pcValidityDate:** Limit date of order validity, if such validity extends to a specific date.
- **pcOrderTime:** Order date/time
- **ExecutedTradeQuantity:** Executed trade quantity.
- **acDisclosedQty:** Disclosed quantity.

The Condition field returns either **occTotal** or **occPartial,** depending on the order status. If trade execution is already underway, cancellation will be partial; otherwise it will be total. In the event of an attempt to cancel a total execution order, MEGA BOLSA will respond with an **ErrorResponse** event containing the Order Not Found message.

2 –FIX Protocol

2.1 FIX

Created in 1992, the Financial Information Protocol (FIX Protocol) is a technical specification for use in trade-related electronic message communication. Jointly developed by banks, brokerage houses, exchanges, public interest companies, industrial associations, institutional investors and Information Technology providers established around the world, FIX is not a software, but rather a global protocol specification for trade-related communication.

The FIX Protocol specification is developed through the voluntary efforts of business technicians and professionals from affiliate companies which together manage the specification. Above and beyond the stringent technical and engineering-related efforts undertaken to ensure ongoing applicability of the protocol to financial market systems, the FIX Protocol benefits from vigorous educational and marketing endeavors aimed at maintaining the specification viable, relevant and attuned to the needs of market participants.

With the purpose of facilitating system updates in line with the FIX Protocol's future upgrades and considering the time saved by not writing a library to work with the FIX Protocol, the QuickFix open source engine was utilized to support FIX Protocol versions 4.2, 4.3 and 4.4 as a multiplatform solution (Windows, Linux, Solaris, FreeBSD and Mac OS X), and its APIs available to C++, Java, NET and Python.

The connection between client and MultiGateway server via FIX occurs in the following manner: The MultiGateway server initializes as Acceptor for each client and sets up the respective TCP/IP port in advance, in LISTEN mode and awaiting a client connection (Initiator).

The client, in turn, connects with this set-up server port. As soon as the connection is established, a FIX message exchange is initialized enabling a client logon process.

A description of FIX messages is shown below, along with their mandatory BOVESPA MultiGateway fields.

2.2 FIX Messages

2.2.1 Header

The FIX message header is in the following format:

Tag	Field name	Required	Remarks
8	Begin String	Yes	Defines the version used by FIX. It should be the first field on the message.
9	BodyLength	Yes	Message length in bytes. It should be the second field on the message.
35	MsgType	Yes	Defines the message type. It should be the third field on the message. Valid values: 0 = Heartbeat 1 = Test Request 2 = Resend Request 3 = Reject 4 = Sequence Reset 5 = Logout

Tag	Field name	Required	Remarks
			8 = Execution Report 9 = Order Cancel Reject A = Logon D = Order – Single F = Order Cancel Request G = Order Cancel/Replace Request
49	SenderCompID	Sim	Defines message sender.
56	TargetCompID	Yes	Defines message recipient.
34	MsgSeqNum	Yes	Sequential number of message.
50	SenderSubID	No	Value used to identify message creator.
142	SenderLocationID	No	Value used to identify message creator location.
57	TargetSubID	No	Value used to identify a specific individual or the only recipient of a message. "ADMIN" reserved for administrator's message, not addressed to a specific user.
143	TargetLocationID	No	Value used to identify message destination location.
97	PossResend	No	It indicates that message may contain information sent under another sequential number. Valid values: Y = Possible resend; N = Original send.
52	SendingTime	Yes	Message send time, always in Universal Time Coordinated (UTC).
122	OrigSendingTime	No	Original message send time, when resend requests are answered, always in Universal Time Coordinated (UTC).
369	LastMsgSeqNumProcessed	No	The last MsgSeqNum received and processed. It can be specified for each sent message.

On messages generated by BOVESPA, the SenderCompID tag should the same as BOVESPACompID, which has the fixed value "*BOVESPA*". As for the SenderSubID and SenderLocationID fields, these will remain the same as BOVESPASubID and BOVESPALocationID, respectively. In this case, the fields TargetCompID, TargetSubID and TargetLocationID fields will be the same as in the client's ID fields which are received at the outset of a session, or on the message of the request that generated the answer.

2.2.2 Trailer

The FIX message ending is in the following format:

Tag	Field name	Required	Remarks
10	CheckSum	Yes	Message length in bytes should be the last field on the message.

2.2.3 Administrator's Messages

The following FIX administrator's messages are supported by BOVESPA MultiGateway:

- Logon
- Logout
- HeartBeat
- TestRequest
- ResendRequest
- Reject
- SequenceReset

2.2.3.1 Logon

The FIX LOGON process can be defined in three parts:

1. The client session (Initiator) establishes communication with the server session (Acceptor).
2. The client sends a Logon message (as described below) to server who in turn should authenticate it and identify it. If client is authenticated, the Logon message is answered, otherwise the connection is terminated;
3. After the Logon message is answered, the client sends a message to sync the sequential numbers.

The FIX Logon message is in the following format:

Tag	Field name	Required	Remarks
98	EncryptMethod	Yes	Valid values: 0 = None / Other
108	HeartBtInt	Yes	HeartBeat interval, defined in seconds.
141	ResetSeqNumFlag	No	It indicates that both sides of the FIX communication should reset the sequential number. Valid values: Y = Yes, sequential number should be reset; N = No, sequential number should not be reset.
35	MsgType	No	Message specific type value is supported. This field is required if the NoMsgTypes field is defined with any value higher that zero.

2.2.3.2 Logout

A standard FIX session logout should consist in an exchange of Logout messages; any other means of program termination should be regarded as abnormal and handled as an error.

Prior to sending a Logout message it is recommended that a TestRequest message be sent to force the HeartBeat on the other side. This helps verification of sequential numbers which must be in sync.

Prior to logging out of a session, the Initiator should wait until the other side answers the Logout message, although the session may be closed if the Acceptor fails to respond within a reasonable time period.

A standard FIX Logout message is in the following format:

Tag	Field name	Required	Remarks
58	Text	No	Text field. **Note:** No maximum length has been determined for this field.

2.2.3.3 HeartBeat

The heartbeat message monitors communication status and helps to maintain message sequence in sync. When both client and server are idle, each side of the connection is expected to send a Heartbeat message to advise each other that the connection is active.

A standard FIX HeartBeat message is in the following format:

Tag	Field name	Required	Remarks
112	TestReqID	No	Identifier used with Test Request messages.

2.2.3.4 TestRequest

A TestRequest message forces the other side to send a heartbeat message. The other application should return the TestReqID field value. BOVESPA MultiGateway fills this space with date and time.

A standard FIX TestRequest message is in the following format:

Tag	Field name	Required	Remarks
112	TestReqID	Yes	Identifier used with Test Request messages.

2.2.3.5 ResendRequest

The ResendRequest message is used to request that messages be resent. It is possible to request one single message, a range of messages or all messages of a given sequence. It is recommended that the reception application store the messages on a first in first out basis. If a message is lost, the subsequent messages should be disregarded and a ResendRequest request be issued.

- To request one single message: BeginSeqNo = EndSeqNo
- To request a range of messages: BeginSeqNo = Beginning, EndSeqNo = End.
- To request all messages: BeginSeqNo = First EndSeqNo = 999999.

A standard FIX ResendRequest message is in the following format:

Tag	Field name	Required	Remarks
7	BeginSeqNo	Yes	Sequential number of first message.
16	EndSeqNo	Yes	Sequential number of last message.

2.2.3.6 Reject

The Reject message should be used to flag that a message has been received, but at least one of its fields is not consistent. Rejected messages are stored and the sequence number increased.

Messages with checksum or size errors will be disregarded. BOVESPA MultiGateway will send a TestRequest request to detect the sequence failure and issue a Resend request. If the error persists for three consecutive times, the message will be disregarded and the sequence value increased.

The Text field will flag the reason for rejection, if the cause has been identified. A standard FIX Reject message is in the following format:

Tag	Field name	Required	Remarks
45	RefSeqNum	Yes	Reference for the sequential number of a message.
371	RefTagID	No	Number of the referenced FIX field.
372	RefMsgType	No	Type of the referenced FIX message.
373	SessionRejectReason	No	Code to identify message rejection: Valid values: 0 = Invalid tag number 1 = Required tag missing 2 = Tag not defined for this message type 3 = Undefined Tag 4 = Tag specified without a value 5 = Value is incorrect (out of range) for this tag 6 = Incorrect data format for value 7 = Decryption problem 8 = Signature problem 9 = CompID problem 10 = SendingTime accuracy problem 11 = Invalid MsgType
58	Text	No	Text field. Note: No maximum length has been determined for this field.
354	EncodedTextLen	No	Length in bytes of the EncodeText field.
355	EncodedText	No	Encoded representation of the Text (58) field contents, following the format defined in the MessageEncoding field.

2.2.3.7 SequenceReset

The sequence reset message is used to reset message sequential numbers. BOVESPA MultiGateway sends a SequenceReset only when a RequestResend infinite loop is detected.

A standard FIX SequenceReset message is in the following format:

Tag	Field name	Required	Remarks
7	BeginSeqNo	Yes	Sequential number of first message.
16	EndSeqNo	Yes	Sequential number of last message.

2.2.4 Application Messages

The following FIX application messages are supported by BOVESPA MultiGateway:

- NewOrderSingle
- OrderCancelRequest

- OrderCancelReject
- ExecutionReport
- OrderCancelReplace

2.2.4.1 NewOrderSingle

The NewOrderSingle message should be sent by a client to register a new order. A standard FIX NewOrderSingle message is in the following format:

Tag	Field name	Required	Remarks
11	ClOrdID	Yes	Single number sent by broker
76	ExecBroker	No	Identifies the executing Broker.
1	Account	Yes	Account, agreed upon between broker and institution. **Note: The user code field should not exceed 8 digits, including the check digit, without a hyphen.**
21	HandInst	Yes	Order handling instruction. Valid values: 1 = Automated execution order, private, no Broker intervention.
111	MaxFloor	No	Maximum number of an order's counterparts that can be displayed at a given time.
55	Symbol	Yes	Stock
54	Side	Yes	Order side. Valid values: 1 = Buy 2 = Sell
60	TransactTime	Yes	Indicates the order's execution/creation time; this field should be filled using the UTC format.
40	OrdType	Yes	Order type. Valid values: 2 = Limit A = On Close K = Market with leftover as limit **See Chapter I, item 3.3**
44	Price:	No	Price per share.
15	Currency	No	\identifies the currency used for the price. BRL, in Brazil's case.
59	TimeInForce	No	Specifies time period that an order remains in force/valid. If this number is missing, it will be interpreted as DAY. Valid values: 0 = Day 1 = Good Till Cancel (GTC) 2 = At the Opening (OPG) 3 = Immediate or Cancel (OC) 4 = Fill or Kill (FOK) 5 = Good Till Crossing (GTX) 6 = Good Till Date (GTD)
432	ExpireDate	No	Order expiration date. Mandatory when 59 = GTD.

Tag	Field name	Required	Remarks
126	ExpireTime	No	Order expiration date and time; this field should be defined in UTC.
120	SettlCurrency	No	Currency code of establishment denomination.
58	Text	No	Text field. **Note:** No maximum length has been determined for this field.
210	MaxShow	No	Maximum number of an order's shares that can be displayed to other clients.
38	OrderQty	Yes	Number of shares requested.

2.2.4.2 OrderCancelRequest

The OrderCancelRequest message should be sent by a client to cancel a previously sent and approved order.
The FIX Cancel Order message is in the following format:

Tag	Field name	Required	Remarks
41	OrigClOrdID	Yes	Single number of an order sent by broker for cancellation. **See Chapter I, item 3.2.2**
37	OrderID	No	Single order identifier indicated by broker.
11	ClOrdID	Yes	Single number sent by broker
1	Account	No	Account, agreed upon between broker and institution.
76	ExecBroker	No	Identifies the executing Broker.
55	Symbol	Yes	Stock symbol.
54	Side	Yes	Order side. Valid values: 1 = Buy 2 = Sell
60	TransactTime	Yes	Indicates the order's execution/creation time; this field should be filled using the UTC format.
38	OrderQty	No	Number of shares requested.
58	Text	No	Text field. **Note:** No maximum length has been determined for this field.

2.2.4.3 OrderCancelReject

The OrderCancelReject message should be sent by MEGA BOLSA with the information that a previous cancellation request cannot be fulfilled.

A standard FIX OrderCancelReject message is in the following format:

Tag	Field name	Required	Remarks
37	OrderID	Yes	Single order identifier provided by broker.
11	ClOrdID	Yes	Single number sent by broker
41	OrigClOrdID	Yes	Single number of an order sent by broker for cancellation.
39	OrdStatus	Yes	Identifier for an order's current status. Valid values: 0 = New 1 = Partially filled 2 = Filled 4 = Cancelled 5 = Replaced 6 = Pending Cancel 8 = Rejected

Tag	Field name	Required	Remarks
			9 = Suspended A = Pending New E = Pending Replace
76	ExecBroker	No	Identifies the executing Broker.
66	ListID	No	Single identifier for list, provided by institution.
1	Account	No	Account, agreed upon between broker and institution.
60	TransactTime	No	Indicates the order's execution/creation time; this field should be filled using the UTC format.
434	CxlRejResponseTo	Yes	Identifies the Cancel Request type in responding. Valid values: 1 - Order Cancel Request 2 - Order Cancel/Replace Request
102	CxlRejReason	No	Code to identify the reason for cancellation rejection. Valid values: 0 = Too late to cancel 1 = Unknown order 2 = Broker Option 3 = Order already in Pending Cancel.
58	Text	No	Text field.
354	EncodedTextLen	No	Length in bytes of the EncodeText field.
355	EncodedText	No	Encoded representation of the Text (58) field contents, following the format defined in the MessageEncoding field.

2.2.4.4 ExecutionReport

The Execution Report message is used to:

1. Confirm order reception;
2. Confirm order modifications;
3. Order status information;
4. Order rejection;

The Execution Report message is in the following format:

Tag	Field name	Required	Remarks
37	OrderID	Yes	Single order identifier provided by broker.
11	ClOrdID	No	Order identifier provided by institution.
41	OrigClOrdID	No	ClOrdID of a previous order, provided by institution and used to identify a previously sent order for cancellation requests.
17	ExecID	Yes	Single message execution identifier, provided by broker (it should be [0] if the ExecTransType field is defined with Status[3])
20	ExecTransType	Yes	Identifies the transaction type. Valid values: 0 = New 1 = Cancel
150	ExecType	Yes	Defines the Execution Report specifically. Valid values:

Tag	Field name	Required	Remarks
			0 = New 1 = Partial fill 2 = Fill 4 = Cancelled 5 = Replace 6 = Pending Cancel 8 = Rejected 9 = Suspended A = Pending New E = Pending Replace
39	OrdStatus	Yes	Identifies an order's current status. Valid values: 0 = New 1 = Partially filled 2 = Filled 4 = Cancelled 5 = Replaced 6 = Pending Cancel 8 = Rejected 9 = Suspended A = Pending New E = Pending Replace
54	Side	Yes	Order side. Valid values: 1 = Buy 2 = Sell
14	CumQty	Yes	Cumulative order executed.
151	LeavesQty	Yes	Quantity pending for a given order.
6	AvgPx	Yes	Average price.
103	OrdRejReason	No	Code to identify the reason for order rejection. Valid values: 0 = Broker option 1 = Unknown symbol 2 = Exchange closed 3 = Order exceeds limit 4 = Too late to enter 5 = Unknown Order 6 = Duplicate Order 7 = Duplicate of a verbally communicated order 8 = Stale Order
1	Account	No	Account, agreed upon between broker and institution.
55	Symbol	Yes	Stock symbol.

Note: Since there is no appropriate tag for trade cancellation – TradeCancel **-** the Text field (tag 58) will be used instead observing the "TCQ=XXXXXXX" standard format, where XXXXXXX is the cancelled trade quantity.

2.2.4.5 OrderCancelReplace

The OrderCancelReplace message should be sent by the client to modify an order. A standard FIX OrderCancelReplace message is in the following format:

Tag	Field name	Required	Remarks
11	ClOrdID	Yes	Single number sent by broker
76	ExecBroker	No	Identifies the executing Broker.
1	Account	Yes	Account, agreed upon between broker and institution. **See Chapter I, item 3.3.**
21	HandInst	Yes	Order handling instruction. Valid values: 1 = Automated execution order, private, no Broker intervention.
111	MaxFloor	No	Maximum number of an order's counterparts that can be displayed at a given time.
55	Symbol	Yes	Stock symbol.
54	Side	Yes	Order side. Valid values: 1 = Buy 2 = Sell
60	TransactTime	Yes	Indicates the order's execution/creation time; this field should be filled using the UTC format.
40	OrdType	Yes	Order type. Valid values: 2 = Limit A = On Close K = Market with leftover as limit **See Chapter I, item 3.3**
15	Currency	No	Identifies the currency used for the price. BRL, in Brazil's case.
59	TimeInForce	No	Specifies time period that an order remains in force/valid. If this number is missing, it will be interpreted as DAY. Valid values: 0 = Day 1 = Good Till Cancel (GTC) 2 = At the Opening (OPG) 3 = Immediate or Cancel (OC) 4 = Fill or Kill (FOK) 5 = Good Till Crossing (GTX) 6 = Good Till Date (GTD)
432	ExpireDate	No	Date when order expires. Mandatory when 59 = GTD.
126	ExpireTime	No	Order expiration date and time; this field should be defined in UTC.
120	SettlCurrency	No	Currency code.
58	Text	No	Text field. **Note:** No maximum length has been determined for this field.
210	MaxShow	No	Maximum number of an order's shares that can be shown to other clients. **See Chapter I, item 3.2.2.**
38	OrderQty	Yes	Quantity of shares requested. **See Chapter I, item 3.2.1.**

3 – Fix Messages

3.1 - NewOrderSingle

3.1.1 Order Registration

The sequence shown below demonstrates an exchange of messages between a client and MultiGateway, illustrating an order registration.

1 – Client sends a NewOrderSingle message.

2 - MultiGateway returns an ExecutionReport message with a PENDING NEW order status, advising it has received the order.

3 - After the order is processed, MultiGateway returns to client an ExecutionReport message with a NEW order status, advising that the order has been registered.

Sequence	1	2	3
Message Received (client)	NewOrderSingle(D)	-	
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300
ClOrdID (11)	FixSample00001	FixSample00001	FixSample00001
CumQty (14)		0	0
ExecID (17)	-	Fix0000000000000000	Fix0000000000000001
ExecTransType (20)	-	0	0
OrderID (37)	-	Fix0000000000000000	1106000001PETR4
OrderQty (38)	100	100	100
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	10	10	10
Side (54)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4
ExecType (150)	-	NEW(0)	NEW(0)
LeavesQty(151)	-	0	100

3.1.2 Order Registration Request followed by Rejection

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration attempt, which is rejected due to a missing mandatory message field.

1 – Client sends a NewOrderSingle message without the Account(1) field.

2 - MultiGateway returns a Reject message with the text “Required tag missing” and a value of 1 in field RefTagID(371) relative to the missing message field.

Sequence	1	2
Message Received (client)	NewOrderSingle(D)	-
Message Sent (multigateway)	-	Reject(3)
Account (1)	-	-
ClOrdID (11)	FixSample00002	-
MsgSeqNum(34)	2	2
OrderQty (38)	100	-
OrdType (40)	LIMIT(2)	LIMIT(2)
Price (44)	10	-
RefSeqNum (45)	-	2
Side (54)	1	-
Symbol (55)	PETR4	-
Text (58)	-	Required tag missing
RefTagID (371)	-	1
RefMsgType (372)	-	NewOrderSingle(D)

SessionRejectReason (373)	-	Required Tag Missing(1)

3.1.3 Freezing of Order

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration attempt followed by a freezing of order message.

1 – Client sends a NewOrderSingle message.

2 - MultiGateway returns an ExecutionReport message with a pending status;

3 – After the order is processed, MultiGateway returns to client an ExecutionReport message with a SUSPEND order status, advising that the order has been frozen.

Sequence	**1**	**2**	**3**
Message Received (client)	NewOrderSingle(D)	-	
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300
ClOrdID (11)	FixSample00003	FixSample00003	FixSample00003
CumQty (14)		0	0
ExecID (17)	-	Fix000000000000002	Fix000000000000003
ExecTransType (20)	-	0	0
OrderID (37)	-	Fix000000000000001	1106000002PETR4
OrderQty (38)	100	100	100
OrdStatus (39)	-	PENDING NEW(A)	SUSPEND(9)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	10	10	10
Side (54)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4
ExecType (150)	-	NEW (0)	SUSPEND(9)
LeavesQty(151)	-	0	100

3.1.4 Freezing of Order followed by Acceptance upon Defreezing

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration followed by a freezing order response and its defreezing.

1 – Client sends a NewOrderSingle message.

2 – MultiGateway returns an ExecutionReport message with a pending status;

3 – After the order is processed, MultiGateway returns to client an ExecutionReport message with a SUSPEND order status, advising that the order has been frozen.

4 – After order acceptance and defreezing, MultiGateway returns to client an ExecutionReport message with a NEW order status, advising that the order has been accepted.

Sequence	**1**	**2**	**3**	**4**
Message Received (client)	NewOrderSingle(D)	-		-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300
ClOrdID (11)	FixSample00004	FixSample00004	FixSample00004	FixSample00004
CumQty (14)		0	0	0
ExecID (17)	-	Fix000000000000004	Fix000000000000005	Fix000000000000006
ExecTransType (20)	-	0	0	0
OrderID (37)	-	Fix000000000000002	1106000003BAHI4	1106000003BAHI4
OrderQty (38)	1000	1000	1000	1000
OrdStatus (39)	-	PENDING NEW(A)	SUSPEND(9)	NEW(0)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	87	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	BAHI4	BAHI4	BAHI4	BAHI4
ExecType (150)	-	NEW(0)	SUSPEND(9)	NEW(0)
LeavesQty(151)	-	0	0	0
Text(58)	-	-	*	-

* This function is prohibited due to the current status of the security.

3.1.5 Freezing of Orders followed by Rejection upon Defreezing

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration followed by a freezing order response and its defreezing.

1 – Client sends a NewOrderSingle message.

2 – MultiGateway returns an ExecutionReport message with a pending status;

3 – After the order is processed, MultiGateway returns to client an ExecutionReport message with a SUSPEND order status, advising that the order has been frozen.

4 – After order acceptance and defreezing, MultiGateway returns to client an ExecutionReport message with a cancelled order status, advising that the order has been rejected.

Sequence	1	2	3	4
Message Received (client)	NewOrderSingle(D)	-		-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300
ClOrdID (11)	FixSample00004	FixSample00004	FixSample00004	FixSample00004
CumQty (14)		0	0	0
ExecID (17)	-	Fix000000000000004	Fix000000000000005	Fix000000000000006
ExecTransType (20)	-	0	0	0
OrderID (37)	-	Fix000000000000002	1106000003BAHI4	1106000003BAHI4
OrderQty (38)	1000	1000	1000	1000
OrdStatus (39)	-	PENDING NEW(A)	SUSPEND(9)	CANCELLED(4)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	87	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	BAHI4	BAHI4	BAHI4	BAHI4
ExecType (150)	-	NEW(0)	SUSPEND(9)	CANCELLED(4)
LeavesQty(151)	-	0	0	1000
Text(58)	-	-	*	TCQ=1000

* This function is prohibited due to the current status of the security.

3.1.6 Order Send followed by Partial Trade, Freezing and Subsequent Defreezing of Order upon Acceptance

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration attempt followed by a partial order response and its freezing.

1 – Client sends a NewOrderSingle message.

2 – MultiGateway returns an ExecutionReport message with a pending status;

3 – After the order is processed, MultiGateway returns to client an ExecutionReport message with a SUSPEND order status, advising that the order has been frozen.

4 – Subsequently, it returns an ExecutionReport message to client with a SUSPENDED order status, advising that the trade was partially executed.

Sequence	1	2	3	4
Message Received (client)	NewOrderSingle(D)	-		-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300
ClOrdID (11)	FixSample00004	FixSample00004	FixSample00004	FixSample00004
CumQty (14)		0	0	1000
ExecID (17)	-	Fix000000000000004	Fix000000000000005	1106000003BAHI401T
ExecTransType (20)	-	0	0	0
OrderID (37)	-	Fix000000000000002	1106000003BAHI4	1106000003BAHI4
OrderQty (38)	3000	3000	3000	3000
OrdStatus (39)	-	PENDING NEW(A)	SUSPEND(9)	SUSPEND(9)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	87	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	BAHI4	BAHI4	BAHI4	BAHI4
ExecType (150)	-	NEW(0)	SUSPEND(9)	PARTIAL FILL (1)
LeavesQty(151)	-	0	0	2000
Text(58)	-	-	*	-

* This function is prohibited due to the current status of the security.

Sequence	3	4
Message Received (client 1)	-	-
Message Sent (multigateway)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300
AvgPx(6)	-	87
ClOrdID (11)	FixSample00004	FixSample00004
CumQty (14)	1000	3000
ExecID (17)	Fix000000000000010	1106000004BAHI402T
ExecTransType (20)	0	0
LastPx(31)	-	87
OrderID (37)	1106000003BAHI4	1106000003BAHI4
OrderQty (38)	3000	3000
OrdStatus (39)	NEW(0)	FILLED(2)
OrdType (40)	LIMIT(2)	LIMIT(2)
Price (44)	87	87
Side (54)	SELL(2)	SELL(2)
Symbol (55)	BAHI4	BAHI4
ExecType (150)	PARTIAL FILL(1)	FILL(2)
LeavesQty(151)	2000	0

Text(58)	-

3.1.7 Freezing of Order followed by Auction

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration attempt followed by a freezing order response and subsequent auction.
1 – Client sends a NewOrderSingle message.
2 – MultiGateway returns an ExecutionReport message with a pending status;
3 – After the order is processed, MultiGateway returns an ExecutionReport message to client with a SUSPEND order status, advising that the order has been frozen.
4 – After freezing of order, MultiGateway returns an ExecutionReport message to client with a NEW order status, advising that the order has been placed in auction.

Sequence	**1**	**2**	**3**	**4**
Message Received (client)	NewOrderSingle(D)	-		-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300
ClOrdID (11)	FixSample00004	FixSample00004	FixSample00004	FixSample00004
CumQty (14)		0	0	0
ExecID (17)	-	Fix0000000000000004	Fix0000000000000005	Fix0000000000000006
ExecTransType (20)	-	0	0	0
OrderID (37)	-	Fix0000000000000002	1106000003BAHI4	1106000003BAHI4
OrderQty (38)	1000	1000	1000	1000
OrdStatus (39)	-	PENDING NEW(A)	SUSPEND(9)	NEW(0)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	87	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	BAHI4	BAHI4	BAHI4	BAHI4
ExecType (150)	-	NEW(0)	SUSPEND(9)	NEW(0)
LeavesQty(151)	-	0	0	0
Text(58)	-	-	*	-

* This function is prohibited due to the current status of security.

3.1.8 New Order Send during Auction

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an order registration attempt while the order is in auction.
1 – Client sends a NewOrderSingle message.
2 – MultiGateway returns an ExecutionReport message with a pending status;
3 – After the order is processed, MultiGateway returns an ExecutionReport message with a NEW order status.

Sequence	**1**	**2**	**4**
Message Received (client)	NewOrderSingle(D)	-	-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300
ClOrdID (11)	FixSample00004	FixSample00004	FixSample00004
CumQty (14)		0	0
ExecID (17)	-	Fix0000000000000004	Fix0000000000000006
ExecTransType (20)	-	0	0
OrderID (37)	-	Fix0000000000000002	1106000003BAHI4
OrderQty (38)	1000	1000	1000
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)
Price (44)	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)

Symbol (55)	BAHI4	BAHI4	BAHI4
ExecType (150)	-	NEW(0)	NEW(0)
LeavesQty(151)	-	0	0
Text(58)	-	-	-

3.2. OrderCancelRequest

3.2.1 Cancellation of a Registered Order

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating the cancellation of a registered order.

1 – Client sends an OrderCancelRequest message;

2 – MultiGateway returns an ExecutionReport message with a PENDING CANCEL order status, advising it has received an order cancellation request.

3 – After the request is processed, MultiGateway returns an ExecutionReport message with a CANCELLED order status, advising that the order has been cancelled.

Sequence	**1**	**2**	**3**
Message Received (client)	OrderCancelRequest(F)	-	
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)
Account (1)	-	300	300
ClOrdID (11)	FixSample00005	FixSample00005	FixSample00005
CumQty (14)	-	0	0
ExecID (17)	-	Fix000000000000007	Fix000000000000008
ExecTransType (20)	-	0	0
OrderID (37)	-	Fix000000000000003	Fix000000000000004
OrderQty (38)	100	100	100
OrdStatus (39)	-	PENDING CANCEL(6)	CANCELLED(4)
OrdType (40)	-	2	2
OrigClOrdID (41)	FixSample00001	FixSample00001	FixSample00001
Price (44)	10	10	10
Side (54)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4
ExecType (150)	-	PENDING CANCEL(6)	CANCELLED(4)
LeavesQty(151)	-	100	0

3.2.2 Cancellation of a Non-Existent Order

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating the cancellation of a non-existent order.

1 – Client sends an OrderCancelRequest message;

2 - MultiGateway analyzes the message and returns an OrderCancelReject message with a REJECTED status and the text “Order not found” to advise that the request was rejected because the specified order was not found.

Sequence	**1**	**2**
Message Received (client)	OrderCancelRequest(F)	-
Message Sent (multigateway)	-	OrderCancelReject(9)
Account (1)	-	300
ClOrdID (11)	FixSample00006	FixSample00006
OrderID (37)	-	Fix000000000000002
OrderQty (38)	100	-
OrdStatus (39)	-	REJECTED(8)
OrigClOrdID (41)	FixSample00099	FixSample00099
Price (44)	10	10
Side (54)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4

Text(58)	-	Order not found
CxlRejResponseTo (434)	-	CancelRequest(1)

3.2.3 Cancellation of a Previously Cancelled Order

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating the cancellation of a previously cancelled order.

1 – Client sends an OrderCancelRequest message;

2 – MultiGateway returns an ExecutionReport message with a PENDING CANCEL order status, advising it has received an order cancellation request.

3 – After the request has been processed, MultiGateway returns to client an ExecutionReport message with a REJECTED order status, with the text "Invalid Operation", advising that the order was rejected because it had been cancelled previously.

Sequence	**1**	**2**	**3**
Message Received (client)	OrderCancelRequest(F)	-	
Message Sent (multigateway)	-	ExecutionReport(8)	OrderCancelReject(9)
Account (1)	-	300	300
ClOrdID (11)	FixSample00007	FixSample00007	FixSample00007
CumQty (14)	-	0	0
ExecID (17)	-	Fix0000000000000009	-
ExecTransType (20)	-	0	-
OrderID (37)	-	Fix0000000000000005	Fix0000000000000006
OrderQty (38)	100	100	-
OrdStatus (39)	-	PENDING CANCEL(6)	REJECTED(8)
OrdType (40)	-	LIMIT(2)	LIMIT(2)
OrigClOrdID (41)	FixSample00001	FixSample00001	FixSample00001
Price (44)	10	10	10
Side (54)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4
Text(58)	-	-	Invalid transaction
ExecType (150)	-	PENDING CANCEL(6)	-
LeavesQty(151)	-	100	-
CxlRejResponseTo (434)	-	-	OrderCancelRequest(1)

3.2.4 Cancellation of an Order at Auction

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating the cancellation of an order during an auction.

1 – Client sends an OrderCancelRequest message;
2 – MultiGateway returns an ExecutionReport message with a pending status;
3 - MultiGateway returns an OrderCancelReject message.

Sequence	1	2	3
Message Received (client)	OrderCancelRequest(F)	-	
Message Sent (multigateway)	-	ExecutionReport(8)	OrderCancelReject(9)
Account (1)	-	300	300
ClOrdID (11)	FixSample00007	FixSample00007	FixSample00007
CumQty (14)	-	0	0
ExecID (17)	-	Fix000000000000009	-
ExecTransType (20)	-	0	-
OrderID (37)	-	Fix000000000000005	Fix000000000000006
OrderQty (38)	1000	1000	-
OrdStatus (39)	-	PENDING CANCEL(6)	REJECTED(8)
OrdType (40)	-	LIMIT(2)	LIMIT(2)
OrigClOrdID (41)	FixSample00001	FixSample00001	FixSample00001
Price (44)	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	BAHI4	BAHI4	BAHI4
ExecType (150)	-	PENDING CANCEL(6)	-
LeavesQty(151)	-	0	-
CxlRejResponseTo (434)	-	-	OrderCancelRequest(1)
Text(58)	-	-	*

* No orders may be cancelled during auction.

3.3 OrderCancelReplace

3.3.1 Modifying a Registered Order

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating modifications made to a registered order.

1 – Client sends an OrderCancelReplace message;
2 – MultiGateway returns an ExecutionReport message with a PENDING REPLACE order status, advising it has received an order modification request.
3 – After the request has been processed, MultiGateway returns an ExecutionReport message with a REPLACED order status, advising that the order has been modified.

Sequence	1	2	3
Message Received (client)	OrderCancelReplace(G)	-	
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)
Account (1)	-	300	300
ClOrdID (11)	FixSample00005	FixSample00005	FixSample00005
CumQty (14)	-	0	0
ExecID (17)	-	Fix0000000000000007	Fix0000000000000008
ExecTransType (20)	-	0	0
OrderID (37)	-	Fix0000000000000003	Fix0000000000000004
OrderQty (38)	100	100	100
OrdStatus (39)	-	PENDING REPLACE(E)	REPLACED(5)
OrdType (40)	-	2	2
OrigClOrdID (41)	FixSample00001	FixSample00001	FixSample00001
Price (44)	10	10	10
Side (54)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4
ExecType (150)	-	PENDING REPLACE(E)	REPLACED(5)
LeavesQty(151)	-	100	100

3.3.2 Changing a Non-Existent Order

The sequence shown below demonstrates an exchange of messages between the client and MultiGateway, illustrating an attempt to modify a non-existent order.

1 – Client sends an OrderCancelReplace message;
2 – MultiGateway analyses the message and returns an OrderCancelReject message with a REJECTED status and the text "Order not found" to advise that the request was rejected because the specified order was not found.

Sequence	1	2
Message Received (client)	OrderCancelReplace(G)	-
Message Sent (multigateway)	-	ExecutionReport (8)
Account (1)	-	300
ClOrdID (11)	FixSample00006	FixSample00006
OrderID (37)	-	Fix0000000000000002
OrderQty (38)	100	-
OrdStatus (39)	-	REJECTED(8)
OrigClOrdID (41)	FixSample00099	FixSample00099
Price (44)	10	10
Side (54)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4
Text(58)	-	Order not found

3.4 Trade

3.4.1 Trade - filled

The sequence shown below demonstrates an exchange of messages between two clients and MultiGateway, illustrating a trade.

1 – Clients 1 and 2 send out NewOrderSingle messages, client 1, a buy message and client 2, a sell message;

2 – MultiGateway returns to both an ExecutionReport message with a PENDING NEW status;

3 – After the request is processed, MultiGateway returns to both clients an ExecutionReport message with a NEW order status, advising that their orders have been registered;

4 – After the orders are matched, the central system advises MultiGateway of the trade and MultiGateway returns to both clients an ExecutionReport message, advising them of the trade.

Sequence	1	2	3	4
Message Received (client 1)	NewOrderSingle(D)	-	-	-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300
AvgPx(6)	-	-	-	21.4
ClOrdID (11)	FixSample00008	FixSample00008	FixSample00008	FixSample00008
CumQty (14)		0	0	100
ExecID (17)	-	Fix000000000000009	Fix000000000000010	1106000004PETR401T
ExecTransType (20)	-	0	0	0
LastPx(31)	-	-	-	21.4
OrderID (37)	-	Fix000000000000007	1106000004PETR4	1106000004PETR4
OrderQty (38)	100	100	100	100
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)	FILLED(2)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	-
Price (44)	10	10	10	10
Side (54)	BUY(1)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4	PETR4
ExecType (150)	-	NEW (0)	NEW(0)	FILL(2)
LeavesQty(151)	-	0	100	0
Text(58)	-	-	-	

Sequence	1	2	3	4
Message Received (client 2)	NewOrderSingle(D)	-	-	-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	500	500	500	500
AvgPx(6)	-	-	-	21.4
ClOrdID (11)	FixSample00009	FixSample00009	FixSample00009	FixSample00009
CumQty (14)		0	0	100
ExecID (17)	-	Fix000000000000011	Fix000000000000012	1106000005PETR401T
ExecTransType (20)	-	0	0	0
LastPx(31)	-	-	-	21.4
OrderID (37)	-	Fix000000000000008	1106000005PETR4	1106000005PETR4
OrderQty (38)	100	100	100	100
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)	FILLED(2)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	-

Price (44)	10	10	10	10
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	PETR4	PETR4	PETR4	PETR4
ExecType (150)	-	NEW (0)	NEW(0)	FILL(2)
LeavesQty(151)	-	0	100	0

3.4.2 Trade – Partially Filled

The sequence shown below demonstrates an exchange of messages between two clients and MultiGateway, illustrating a partially filled trade.

1 – Clients 1 and 2 send NewOrderSingle messages, client 1, a buy order for a quantity of 20 and client 2 a sell order for a quantity of 15;

2 – MultiGateway returns to both an ExecutionReport message with a PENDING NEW status;

3 – After the request is processed, MultiGateway returns to both clients an ExecutionReport message with a NEW order status, advising that their orders have been registered;

4 – After the orders have been matched, the central system advises MultiGateway of the trade and MultiGateway then returns to client 1 an ExecutionReport message advising him/her of the partial trade, with pending actions, and an ExecutionReport trade message to client 2.

Sequence	**1**	**2**	**3**	**4**
Message Received (client 1)	NewOrderSingle(D)	-	-	-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300
AvgPx(6)	-	-	-	20
ClOrdID (11)	FixSample00011	FixSample00011	FixSample00011	FixSample00011
CumQty (14)		0	0	15
ExecID (17)	-	Fix000000000000013	Fix000000000000014	1106000006PETR402T
ExecTransType (20)	-	0	0	0
LastPx(31)	-	-	-	20
OrderID (37)	-	Fix000000000000009	1106000006PETR4	1106000006PETR4
OrderQty (38)	20	20	20	20
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)	PARTIALLY FILLED(1)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	-
Price (44)	20	20	20	20
Side (54)	BUY(1)	BUY(1)	BUY(1)	BUY(1)
Symbol (55)	PETR4	PETR4	PETR4	PETR4
ExecType (150)	-	NEW (0)	NEW(0)	PARTIAL FILL(1)
LeavesQty(151)	-	0	20	5

Sequence	**1**	**2**	**3**	**4**
Message Received (client 2)	NewOrderSingle(D)	-	-	-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	500	500	500	500
AvgPx(6)	-	-	-	20
ClOrdID (11)	FixSample00012	FixSample00012	FixSample00012	FixSample00012
CumQty (14)		0	0	15
ExecID (17)	-	Fix000000000000015	Fix000000000000016	1106000007PETR402T
ExecTransType (20)	-	0	0	0
LastPx(31)	-	-	-	20

OrderID (37)	-	Fix000000000000010	1106000007PETR4	1106000007PETR4
OrderQty (38)	15	15	15	15
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)	FILLED(2)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	-
Price (44)	20	20	20	20
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Symbol (55)	PETR4	PETR4	PETR4	PETR4
ExecType (150)	-	NEW (0)	NEW(0)	FILL(2)
LeavesQty(151)	-	0	15	0

3.4.3 Cancel Trade

The sequence shown below demonstrates an exchange of messages between a client and MultiGateway, illustrating the cancellation of a trade.

1 – Client sends a NewOrderSingle sell order message.
2 – MultiGateway returns an ExecutionReport message with a PENDING NEW status;
3 – After the request has been processed, MultiGateway returns an ExecutionReport message with a NEW order status, advising that the order has been registered;
4 – After the orders are matched, the central system advises MultiGateway of the trade and MultiGateway returns to the client an ExecutionReport message, advising him/her of the trade.
5 – After the trade has been cancelled, the central system advises MultiGateway of the cancellation and MultiGateway returns to the clients an ExecutionReport message with a NEW status, advising the clients of the trade cancellation.

Sequence	**1**	**2**	**3**	**4**	**5**
Message Received (client 1)	NewOrderSingle(D)	-	-	-	-
Message Sent (multigateway)	-	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)	ExecutionReport(8)
Account (1)	300	300	300	300	300
AvgPx(6)	-	-	-	87	0
ClOrdID (11)	FixSample00004	FixSample00004	FixSample00004	FixSample00004	FixSample00004
CumQty (14)		0	0	1000	0
ExecID (17)	-	Fix000000000000017	Fix000000000000018	1106000008BAHI04T	1106000008BAHI401C
ExecRefID	-	-	-	-	1106000008BAHI404T
ExecTransType (20)	-	0	0	0	0
LastPx(31)	-	-	-	87	87
LastQty(32)	-	-	-	-	1000
OrderID (37)	-	Fix000000000000011	1106000008BAHI4	1106000008BAHI4	1106000008BAHI4
OrderQty (38)	1000	1000	1000	1000	1000
OrdStatus (39)	-	PENDING NEW(A)	NEW(0)	FILLED(2)	CANCELLED(4)
OrdType (40)	LIMIT(2)	LIMIT(2)	LIMIT(2)	-	LIMIT(2)
Price (44)	87	87	87	87	87
Side (54)	SELL(2)	SELL(2)	SELL(2)	SELL(2)	SELL(2)
Text (58)	-	-	-	-	TCQ=1000
Symbol (55)	BAHI4	BAHI4	BAHI4	BAHI4	BAHI4
ExecType (150)	-	NEW (0)	NEW(0)	FILL(2)	CANCELLED(4)
LeavesQty(151)	-	0	1000	0	0